Linux Professional Institute

Linux Essentials

Versão 1.6
Português

010

Table of Contents

TÓPICO 1: A COMUNIDADE LINUX E A CARREIRA OPEN SOURCE . . . **1**
1.1 A evolução do Linux e sistemas operacionais populares . . . **2**
1.1 Lição 1 . . . 3
Introdução . . . 3
Distribuições . . . 4
Sistemas embarcados . . . 5
O Linux e a Nuvem . . . 7
Exercicios Guiados . . . 8
Exercícios Exploratórios . . . 9
Resumo . . . 10
Respostas aos Exercícios Guiados . . . 11
Respostas aos Exercícios Exploratórios . . . 13
1.2 Principais Aplicações Open Source . . . **14**
1.2 Lição 1 . . . 15
Introdução . . . 15
Pacotes de software . . . 15
Instalação do pacote . . . 16
Remoção de Pacotes . . . 19
Aplicativos de escritório . . . 21
Navegadores web . . . 22
Multimídia . . . 22
Programas de servidor . . . 23
Compartilhamento de dados . . . 24
Administração de redes . . . 26
Linguagens de programação . . . 27
Exercícios Guiados . . . 30
Exercícios Exploratórios . . . 32
Sumário . . . 33
Respostas aos Exercícios Guiados . . . 34
Respostas aos Exercícios Exploratórios . . . 36
1.3 Entendendo o Software Open Source e suas Licenças . . . **37**
1.3 Lição 1 . . . 38
Introdução . . . 38
Definição de Software Livre e de Código Aberto . . . 38
Licenças . . . 41
Modelos de negócios em Open Source . . . 45
Exercícios Guiados . . . 48
Exercícios Exploratórios . . . 49

Sumário . . . 50
Respostas aos Exercícios Guiados . . . 51
Respostas aos Exercícios Exploratórios . . . 52
1.4 ICT Habilidades ICT e trabalhando no Linux . . . 54
1.4 Lição 1 . . . 55
Introdução . . . 55
Interfaces de usuário Linux . . . 56
Usos do Linux na indústria . . . 58
Problemas de privacidade ao se usar a internet . . . 59
Criptografia . . . 63
Exercícios Guiados . . . 66
Exercícios Exploratórios . . . 68
Sumário . . . 69
Respostas aos Exercícios Guiados . . . 70
Respostas aos Exercícios Exploratórios . . . 72
TÓPICO 2: ENCONTRANDO SEU CAMINHO EM UM SISTEMA LINUX . . . 73
2.1 O básico sobre a linha de comando . . . 74
2.1 Lição 1 . . . 75
Introdução . . . 75
Estrutura da linha de comando . . . 77
Tipos de comportamento dos comandos . . . 78
Citação . . . 78
Exercícios Guiados . . . 82
Exercícios Exploratórios . . . 84
Resumo . . . 85
Respostas aos Exercícios Guiados . . . 86
Respostas aos Exercícios Exploratórios . . . 87
2.1 Lição 2 . . . 88
Introdução . . . 88
Variáveis . . . 88
Manipulação de variáveis . . . 89
Exercícios Guiados . . . 94
Exercícios Exploratórios . . . 95
Sumário . . . 96
Respostas aos Exercícios Guiados . . . 97
Respostas aos Exercícios Exploratórios . . . 98
2.2 O Usando a linha de comando para conseguir ajuda . . . 100
2.2 Lição 1 . . . 101
Introdução . . . 101
Como obter ajuda na linha de comando . . . 101

Como localizar arquivos . . . 104
Exercícios Guiados . . . 107
Exercícios Exploratórios . . . 109
Resumo . . . 110
Respostas aos Exercícios Guiados . . . 111
Respostas aos Exercícios Exploratórios . . . 114
2.3 Usando diretórios e listando arquivos . . . 116
2.3 Lição 1 . . . 117
Introdução . . . 117
Arquivos e diretórios . . . 117
Nomes de arquivos e diretórios . . . 118
Navegando no sistema de arquivos . . . 118
Caminhos absolutos e relativos . . . 120
Exercícios Guiados . . . 122
Exercícios Exploratórios . . . 124
Resumo . . . 125
Respostas aos Exercícios Guiados . . . 126
Respostas aos Exercícios Exploratórios . . . 129
2.3 Lição 2 . . . 130
Introdução . . . 130
Diretórios Home . . . 130
O caminho relativo especial para home . . . 132
Caminhos de arquivos relativos-a-home . . . 133
Arquivos e diretórios ocultos . . . 134
A opção de lista longa . . . 135
Opções adicionais de ls . . . 136
A recursão no Bash . . . 136
Exercícios Guiados . . . 139
Exercícios Exploratórios . . . 141
Resumo . . . 142
Respostas aos Exercícios Guiados . . . 143
Respostas aos Exercícios Exploratórios . . . 145
2.4 Criando, Movendo e Deletando Arquivos . . . 146
2.4 Lição 1 . . . 147
Introdução . . . 147
Maiúsculas e minúsculas . . . 148
Criando diretórios . . . 148
Criando arquivos . . . 150
Renomeando arquivos . . . 151
Movendo arquivos . . . 152

Excluindo arquivos e diretórios . . . 153
Copiando arquivos e diretórios . . . 155
Globbing . . . 157
Exercícios Guiados . . . 162
Exercícios Exploratórios . . . 164
Resumo . . . 165
Respostas aos Exercícios Guiados . . . 167
Respostas aos Exercícios Exploratórios . . . 170
TÓPICO 3: O PODER DA LINHA DE COMANDO . . . 172
3.1 Empacotando arquivos na linha de comando . . . 173
3.1 Lição 1 . . . 174
Introdução . . . 174
Ferramentas de compressão . . . 175
Ferramentas de empacotamento . . . 178
Gerenciando arquivos ZIP . . . 181
Exercícios Guiados . . . 183
Exercícios Exploratórios . . . 184
Resumo . . . 185
Respostas aos Exercícios Guiados . . . 187
Respostas aos Exercícios Exploratórios . . . 189
3.2 Pesquisando e extraindo dados de arquivos . . . 190
3.2 Lesson 1 . . . 191
Introdução . . . 191
Redirecionamento de E/S . . . 191
Pipes na linha de comando . . . 196
Exercícios Guiados . . . 199
Exercícios Exploratórios . . . 200
Resumo . . . 201
Respostas aos Exercícios Guiados . . . 202
Respostas aos Exercícios Exploratórios . . . 204
3.2 Lição 2 . . . 205
Introdução . . . 205
Pesquisando dentro de arquivos com `grep` . . . 205
Expressões regulares . . . 206
Exercícios Guiados . . . 210
Exercícios Exploratórios . . . 211
Resumo . . . 212
Respostas aos Exercícios Guiados . . . 213
Respostas aos Exercícios Exploratórios . . . 215
3.3 Transformando comandos em Scripts . . . 217

3.3 Lesson 1 . . . 218
Introdução . . . 218
Exibindo a saída . . . 218
Tornando um script executável . . . 219
Comandos e PATH . . . 219
Permissões de execução . . . 220
Definindo o intérprete . . . 221
Variáveis . . . 222
Usando aspas com variáveis . . . 224
Argumentos . . . 225
Retorno do número de argumentos . . . 227
Lógica Condicional . . . 227
Exercícios guiados . . . 229
Exercícios Exploratórios . . . 231
Resumo . . . 232
Respostas aos Exercícios Guiados . . . 234
Respostas aos Exercícios Exploratórios . . . 236
3.3 Lição 2 . . . 238
Introdução . . . 238
Códigos de saída . . . 239
Como manipular múltiplos argumentos . . . 241
Loops for . . . 242
Verificando erros com expressões regulares . . . 245
Exercícios guiados . . . 247
Exercícios Exploratórios . . . 249
Resumo . . . 250
Respostas aos Exercícios Guiados . . . 251
Respostas aos Exercícios Exploratórios . . . 253
TÓPICO 4: O SISTEMA OPERACIONAL LINUX . . . 254
4.1 Escolhendo um Sistema Operacional . . . 255
4.1 Lição 1 . . . 256
Introdução . . . 256
O que é um sistema operacional . . . 256
Escolhendo uma distribuição Linux . . . 257
Sistemas operacionais não-Linux . . . 261
Exercícios Guiados . . . 264
Exercícios Exploratórios . . . 266
Resumo . . . 267
Respostas aos Exercícios Guiados . . . 268
Respostas aos Exercícios Exploratórios . . . 270

4.2 Entendendo o Hardware do Computador **271**
4.2 Lição 1 272
Introdução 272
Fontes de alimentação 273
Placa-mãe 273
Memória 274
Processadores 275
Armazenamento 278
Partições 279
Periféricos 280
Drivers e arquivos de dispositivo 281
Exercícios Guiados 283
Exercícios Exploratórios 284
Resumo 285
Respostas aos Exercícios Guiados 286
Respostas aos Exercícios Exploratórios 288
4.3 Onde os dados são armazenados **289**
4.3 Lição 1 290
Introdução 290
Programas e suas configurações 291
O kernel do Linux 295
Dispositivos de hardware 298
Memória e tipos de memória 300
Exercícios Guiados 303
Exercícios Exploratórios 305
Resumo 306
Respostas aos Exercícios Guiados 308
Respostas aos Exercícios Exploratórios 310
4.3 Lição 2 311
Introdução 311
Processos 311
Registro do sistema e mensagens do sistema 315
Exercícios Guiados 321
Exercícios Exploratórios 324
Resumo 326
Respostas aos Exercícios Guiados 328
Respostas aos Exercícios Exploratórios 332
4.4 Seu Computador na Rede **334**
4.4 Lição 1 335
Introdução 335

Comunicação na camada de link . . . 336
Rede IPv4 . . . 337
Rede IPv6 . . . 342
DNS . . . 345
Sockets . . . 347
Exercícios Guiados . . . 349
Exercícios Exploratórios . . . 350
Resumo . . . 351
Respostas aos Exercícios Guiados . . . 352
Respostas aos Exercícios Exploratórios . . . 353
TÓPICO 5: SEGURANÇA E PERMISSÕES DE ARQUIVOS . . . 355
5.1 Segurança Básica e Identificação de Tipos de Usuários . . . 356
5.1 Lição 1 . . . 357
Introdução . . . 357
Contas . . . 358
Obtendo informações sobre os usuários . . . 361
Troca de usuário e aumento de privilégios . . . 363
Arquivos de controle de acesso . . . 364
Exercícios Guiados . . . 372
Exercícios Exploratórios . . . 374
Resumo . . . 375
Respostas aos Exercícios Guiados . . . 377
Respostas aos Exercícios Exploratórios . . . 379
5.2 Criando Usuários e Grupos . . . 381
5.2 Lição 1 . . . 382
Introdução . . . 382
O Arquivo `/etc/passwd` . . . 383
O arquivo `/etc/group` . . . 384
O arquivo `/etc/shadow` . . . 384
O arquivo `/etc/gshadow` . . . 385
Adicionando e removendo contas de usuário . . . 386
O diretório de esqueleto . . . 388
Adicionando e excluindo grupos . . . 389
O comando `passwd` . . . 389
Exercícios Guiados . . . 391
Exercícios Exploratórios . . . 393
Resumo . . . 394
Respostas aos Exercícios Guiados . . . 395
Respostas aos Exercícios Exploratórios . . . 397
5.3 Gerenciando permissões e donos de arquivos . . . 400

5.3 Lição 1 . . . 401
Introdução . . . 401
Como consultar informações sobre arquivos e diretórios . . . 401
E os diretórios? . . . 403
Exibindo arquivos ocultos . . . 403
Compreendendo os tipos de arquivos . . . 404
Entendendo as permissões . . . 405
Modificando as permissões de arquivo . . . 407
Modo simbólico . . . 408
Modo numérico . . . 410
Modificando o proprietário de um arquivo . . . 410
Consultando os grupos . . . 411
Permissões especiais . . . 412
Exercícios Guiados . . . 416
Exercícios Exploratórios . . . 418
Resumo . . . 419
Respostas aos Exercícios Guiados . . . 420
Respostas aos Exercícios Exploratórios . . . 423
5.4 Diretórios e arquivos especiais . . . 426
5.4 Lição 1 . . . 427
Introdução . . . 427
Arquivos temporários . . . 427
Compreendendo os links . . . 429
Exercícios Guiados . . . 434
Exercícios Exploratórios . . . 435
Resumo . . . 438
Respostas aos Exercícios Guiados . . . 439
Respostas aos Exercícios Exploratórios . . . 440
Imprint . . . 444

Linux Professional Institute

Tópico 1: A comunidade Linux e a carreira Open Source

1.1 A evolução do Linux e sistemas operacionais populares

Referência ao LPI objectivo

Linux Essentials version 1.6, Exam 010, Objective 1.1

Peso

2

Áreas chave de conhecimento

- Distribuições
- Sistemas Embarcados
- Linux na Nuvem

Segue uma lista parcial dos arquivos, termos e utilitários utilizados

- Debian, Ubuntu (LTS)
- CentOS, openSUSE, Red Hat, SUSE
- Linux Mint, Scientific Linux
- Raspberry Pi, Raspbian
- Android

1.1 Lição 1

Certificação:	Linux Essentials
Versão:	1.6
Tópico:	1 A Comunidade Linux e uma carreira em open source
Objetivo:	1.1 A evolução do Linux e os sistemas operacionais populares
Lição:	1 de 1

Introdução

O Linux é um dos sistemas operacionais mais populares. Linus Torvalds começou a desenvolvê-lo em 1991 inspirado no Unix, um outro sistema operacional criado nos anos 70 pelos Laboratórios AT&T. O Unix foi projetado para computadores pequenos. Na época, eram consideradas “pequenas” as máquinas que não precisavam de um cômodo inteiro com ar condicionado e que custavam menos de um milhão de dólares. Mais tarde, essa categoria passou a designar as máquinas que podiam ser erguidas por duas pessoas. Até então, o Unix não estava disponível para pequenos computadores, como os computadores de escritório, que eram baseados na plataforma x86. Assim, Linus, na época um estudante, começou a implementar um sistema operacional semelhante ao Unix, mas capaz de rodar nessa plataforma.

O Linux, em sua maior parte, lança mão dos mesmos princípios e idéias básicas do Unix, mas não contém código Unix, pois trata-se de um projeto independente. O Linux não é suportado por uma empresa individual, mas por uma comunidade internacional de programadores. Disponível gratuitamente, pode ser usado por qualquer pessoa, sem restrições.

Distribuições

Uma *distribuição* Linux é um pacote que consiste em um *kernel* Linux, mais uma seleção de aplicativos mantidos por uma empresa ou comunidade de usuários. O objetivo de uma distribuição é otimizar o kernel e os aplicativos que rodam no sistema operacional para um determinado tipo de uso ou grupo de usuários. As distribuições freqüentemente incluem ferramentas próprias para a instalação de software e administração do sistema. Por essa razão, certas distribuições são usadas principalmente em ambientes desktop, por serem mais fáceis de usar, enquanto outras são mais comumente instaladas em servidores para usar os recursos disponíveis da maneira mais eficiente possível.

Outra maneira de classificar as distribuições é de acordo com a *família de distribuições* a que pertencem. As distribuições da família Debian usam o gerenciador de pacotes `dpkg` para gerenciar o software executado no sistema operacional. Os pacotes que podem ser instalados com o gerenciador de pacotes são mantidos por membros voluntários da comunidade da distribuição. Os mantenedores usam o formato de pacote `deb` para especificar como o software é instalado no sistema operacional e como é configurado por padrão. Assim como uma distribuição, um pacote é constituído por um conjunto de programas, mais uma configuração e documentação correspondentes, facilitando a instalação, atualização e utilização do software pelo usuário.

O *Debian GNU/Linux* é a maior distribuição da família Debian. O Projeto Debian GNU/Linux foi lançado por Ian Murdock em 1993 e conta, hoje, com milhares de voluntários trabalhando no projeto. O objetivo do Debian GNU/Linux é fornecer um sistema operacional extremamente confiável. Ele também promove a visão de Richard Stallman de um sistema operacional que respeite as liberdades do usuário de executar, estudar, distribuir e aprimorar o software. Por essa razão, ele também não fornece nenhum programa proprietário por padrão.

O *Ubuntu* é outra distribuição baseada no Debian que merece ser mencionada. O Ubuntu foi criado por Mark Shuttleworth e sua equipe em 2004, com a missão de produzir um ambiente desktop Linux fácil de usar. A ideia do Ubuntu é fornecer um software livre para todas as pessoas ao redor do mundo, bem como reduzir o custo dos serviços profissionais. A distribuição lança uma nova versão a cada seis meses, além de uma versão com suporte de longo prazo a cada 2 anos.

O *Red Hat* é uma distribuição Linux desenvolvida e mantida pela empresa de software de mesmo nome, adquirida pela IBM em 2019. A distribuição Red Hat Linux foi iniciada em 1994 e rebatizada em 2003 como *Red Hat Enterprise Linux*, ou RHEL. É fornecida às empresas como uma solução empresarial confiável suportada pela Red Hat e vem com software destinado a facilitar o uso do Linux em ambientes de servidores profissionais. Alguns de seus componentes requerem assinaturas ou licenças pagas. O projeto *CentOS* usa o código-fonte livre do Red Hat Enterprise Linux para compilar uma distribuição inteiramente gratuita, mas que não conta com o serviço de

suporte comercial.

Tanto o RHEL quanto o CentOS são otimizados para uso em ambientes de servidor. O projeto *Fedora* foi fundado em 2003 com a ideia de criar uma distribuição Linux voltada para computadores desktop. A Red Hat iniciou e ainda mantém essa distribuição. O Fedora é muito progressista e adota novas tecnologias rapidamente, sendo frequentemente considerado um banco de testes para novas tecnologias que, mais tarde, poderão ser incluídas no RHEL. Todas as distribuições baseadas em Red Hat usam o formato de pacote `rpm`.

A empresa SUSE foi fundada em 1992 na Alemanha como um provedor de serviços Unix. A primeira versão do *SUSE Linux* foi lançada em 1994. Ao longo dos anos, o SUSE Linux tornou-se mais conhecido por sua ferramenta de configuração YaST, que permite aos administradores instalar e configurar software e hardware, configurar servidores e redes. Semelhante ao RHEL, o SUSE oferece o *SUSE Linux Enterprise Server*, sua edição comercial, com lançamentos menos frequentes e adequado para a implantação em empresas e ambientes de produção. É distribuído como um servidor, bem como um ambiente desktop, com pacotes adequados para fins específicos. Em 2004, a SUSE lançou o projeto *openSUSE*, que permitia que os desenvolvedores e usuários testassem e desenvolvessem ainda mais o sistema. A distribuição openSUSE pode ser baixada gratuitamente.

Muitas distribuições independentes foram lançadas ao longo dos anos. Algumas delas se baseiam em Red Hat ou Ubuntu, outras são projetadas para aprimorar uma propriedade específica de um sistema ou hardware. Existem distribuições construídas com funcionalidades específicas, como o *QubesOS*, um ambiente de desktop extremamente seguro, ou o *Kali Linux*, que oferece um ambiente para explorar vulnerabilidades de software e é usado principalmente para testes de intrusão. Recentemente, diversas distribuições Linux minúsculas foram projetadas para rodar especificamente em containers Linux, como o Docker. Existem também distribuições construídas especificamente para componentes de sistemas embarcados e mesmo dispositivos inteligentes.

Sistemas embarcados

Os sistemas embarcados são uma combinação de hardware e software projetados para cumprir uma função específica dentro de um sistema maior. Normalmente fazem parte de outros dispositivos e ajudam a controlá-los. Podem ser encontrados em aplicações automotivas, médicas e até militares. Devido a essa ampla variedade de aplicações, Uma variedade de sistemas operacionais baseados no kernel do Linux foi desenvolvida para uso em sistemas embarcados. uma parte significativa dos dispositivos inteligentes usa um sistema operacional baseado no kernel do Linux.

Assim, em sistemas embarcados temos software embarcado, cujo objetivo é acessar o hardware e torná-lo utilizável. Dentre as principais vantagens do Linux sobre qualquer software embarcado

proprietário estão a compatibilidade entre plataformas de diferentes fornecedores, desenvolvimento, suporte e ausência de taxas de licença. Dois dos mais populares projetos de software embarcado são o Android, usado principalmente em telefones celulares por diferentes fabricantes, e o Raspbian, que é usado principalmente no Raspberry Pi.

Android

O Android é um sistema operacional móvel desenvolvido principalmente pelo Google. A Android Inc. foi fundada em 2003 em Palo Alto, Califórnia. A empresa inicialmente criou um sistema operacional destinado a rodar em câmeras digitais. Em 2005, o Google comprou a Android Inc. e transformou esse sistema em um dos maiores sistemas operacionais móveis.

A base do Android é uma versão modificada do kernel do Linux junto com outros softwares de código aberto. O sistema foi desenvolvido principalmente para dispositivos touchscreen, mas o Google desenvolveu também versões para smart TVs e smartwatches. Diferentes versões do Android foram desenvolvidas para consoles de jogos, câmeras digitais e mesmo PCs.

O código do Android está disponível gratuitamente no *Android Open Source Project* (AOSP). O Google oferece uma série de componentes proprietários juntamente com o núcleo de código aberto do Android. Dentre esses componentes estão aplicativos como Google Agenda, Google Maps, Google Mail, o navegador Chrome e a Google Play Store, que facilita a instalação de aplicativos. A maioria dos usuários considera essas ferramentas como parte integrante da sua experiência no Android. Assim, praticamente todos os dispositivos móveis na Europa e na América que vêm com Android instalado incluem software proprietário do Google.

O Android oferece muitas vantagens em dispositivos embarcados. O sistema operacional é intuitivo e fácil de usar, graças à interface gráfica de usuário, e conta com uma comunidade de desenvolvedores muito ampla, sendo fácil encontrar ajuda para o desenvolvimento. Ele também é suportado pela maioria dos fabricantes de hardware com um driver Android, sendo assim fácil e econômico criar protótipos de um sistema inteiro.

O Raspbian e o Raspberry Pi

O Raspberry Pi é um computador de baixo custo, do tamanho de um cartão de crédito, que pode exercer o papel de um computador de mesa inteiramente funcional, mas também pode ser usado dentro de um sistema Linux embarcado. É desenvolvido pela Raspberry Pi Foundation, uma instituição educacional de caridade no Reino Unido. Sua finalidade principal é ensinar os jovens a programar e compreender a funcionalidade dos computadores. O Raspberry Pi pode ser projetado e programado para realizar tarefas e operações que fazem parte de um sistema muito mais complexo.

Dentre as particularidades do Raspberry Pi temos um conjunto de pinos GPIO (General Purpose Input-Output, ou entrada e saída de uso geral) que podem ser usados para conectar dispositivos eletrônicos e placas de expansão. Isso permite usar o Raspberry Pi como uma plataforma para desenvolvimento de hardware. Embora tenha sido criado para fins educacionais, os Raspberry Pis são utilizados atualmente em diversos projetos caseiros, bem como para prototipagem industrial no desenvolvimento de sistemas embarcados.

O Raspberry Pi emprega processadores ARM. Muitos sistemas operacionais, incluindo o Linux, rodam no Raspberry Pi. Como o Raspberry Pi não contém um disco rígido, o sistema operacional é iniciado a partir de um cartão de memória SD. Uma das distribuições Linux mais importantes para o Raspberry Pi é o *Raspbian*. Como o nome sugere, ele pertence à família de distribuição Debian. Ele é personalizado para ser instalado no hardware do Raspberry Pi e oferece mais de 35000 pacotes otimizados para esse ambiente. Além do Raspbian, existem várias outras distribuições Linux para o Raspberry Pi, como por exemplo o Kodi, que transforma o Raspberry Pi numa central de mídia.

O Linux e a Nuvem

O termo *cloud computing*, ou computação em nuvem, refere-se a uma forma padronizada de consumir recursos de computação, seja comprando-os de um provedor público de nuvem ou criando uma nuvem privada. Segundo relatórios de 2017, o Linux está por trás de 90% da carga de trabalho das nuvens públicas. Todos os provedores de serviços de nuvem, do *Amazon Web Services* (AWS) ao *Google Cloud Platform* (GCP), oferecem diferentes formas de Linux. Mesmo a Microsoft hoje oferece máquinas virtuais baseadas em Linux em sua nuvem *Azure*.

O Linux é comumente incluído nas ofertas de *Infraestrutura como Serviço* (IaaS). As instâncias de IaaS são máquinas virtuais provisionadas em poucos minutos na nuvem. Ao iniciar uma instância de IaaS, uma imagem contendo os dados é escolhida e disponibilizada para a nova instância. Os provedores de nuvem oferecem diversas imagens que contêm instalações de distribuições populares de Linux prontas para executar, além de suas próprias versões de Linux. O usuário de nuvem escolhe a imagem que contém sua distribuição preferida e ganha acesso quase imediato a uma instância da nuvem que roda aquela distribuição. A maioria dos provedores de nuvem adiciona ferramentas às imagens para adaptar a instalação a uma instância específica de nuvem. Essas ferramentas servem, por exemplo, para estender os sistemas de arquivo da imagem de modo a que se encaixem perfeitamente no disco rígido real da máquina virtual.

Exercicios Guiados

1. Quais as diferenças entre o Debian GNU/Linux e o Ubuntu? Cite dois aspectos.

2. Em quais os ambientes/plataformas o Linux é mais usado? Cite três ambientes/plataformas diferentes e dê o nome de uma distribuição que pode ser usada em cada um deles.

3. Você planeja instalar uma distribuição de Linux em um novo ambiente. Cite quatro coisas a se considerar na escolha de uma distribuição.

4. Cite três dispositivos que rodam Android OS, além de smartphones.

5. Explique três grandes vantagens da computação em nuvem.

Exercícios Exploratórios

1. Levando em conta o custo e o desempenho, quais as distribuições mais adequadas para uma empresa que tem como objetivo reduzir custos com licenças sem prejudicar o desempenho? Justifique sua resposta.

2. Quais são as principais vantagens do Raspberry Pi e quais funções ele pode realizar em um contexto empresarial?

3. Quais distribuições são propostas pelo Amazon Cloud Services e pelo Google Cloud? Cite pelo menos três mais comuns e duas diferentes.

Resumo

Nesta lição, você aprendeu:

- Quais distribuições de Linux existem
- O que são sistemas com Linux embarcado
- Como são usados os sistemas com Linux embarcado
- Diferentes aplicações do Android
- Diferentes usos de um Raspberry Pi
- O que é a Computação em Nuvem
- Qual o papel do Linux na computação em nuvem

Respostas aos Exercícios Guiados

1. Quais as diferenças entre o Debian GNU/Linux e o Ubuntu? Cite dois aspectos.

   O Ubuntu baseia-se em um instantâneo do Debian, e por isso existem muitas semelhanças entre eles. Todavia, eles não deixam de apresentar diferenças significativas. A primeira delas seria a usabilidade para os iniciantes. O Ubuntu é recomendado aos iniciantes devido à sua facilidade de uso; já o Debian é recomendado para usuários mais avançados. A principal diferença reside na complexidade da configuração de usuário, que o Ubuntu não exige durante o processo de instalação.

   Outra diferença seria a estabilidade de cada distribuição. O Debian é considerado mais estável em comparação com o Ubuntu. Isso acontece porque ele recebe menos atualizações, que são testadas em detalhe, e o sistema operacional como um todo tem mais estabilidade. Por outro lado, o Ubuntu permite que o usuário utilize os lançamentos mais recentes de software e todas as novas tecnologias.

2. Em quais os ambientes/plataformas o Linux é mais usado? Cite três ambientes/plataformas diferentes e dê o nome de uma distribuição que pode ser usada em cada um deles.

   Dentre os ambientes/plataformas mais comuns poderíamos citar smartphone, desktop e servidor. Nos smartphones, ele pode ser empregado em distribuições como o Android. Para desktop e servidores, é possível escolher qualquer distribuição que pareça adequada à funcionalidade da máquina em questão, como Debian ou Ubuntu, ou ainda CentOS e Red Hat Enterprise Linux.

3. Você planeja instalar uma distribuição de Linux em um novo ambiente. Cite quatro coisas a se considerar na escolha de uma distribuição.

   Ao escolher uma distribuição, alguns elementos importantes a considerar são custo, desempenho, escalabilidade, estabilidade e as exigências de hardware do sistema.

4. Cite três dispositivos que rodam Android OS, além de smartphones.

   Dentre os outros dispositivos que usam Android estão as smart TVs, tablets, Android Auto e smartwatches.

5. Explique três grandes vantagens da computação em nuvem.

   As principais vantagens da computação em nuvem são flexibilidade, facilidade de recuperação e baixo custo. Os serviços baseados na nuvem são fáceis de implementar e escalar, dependendo das necessidades da empresa. Eles são mais vantajosos para as soluções de backup e

recuperação, já que permitem que uma empresa se recupere rapidamente e sem grandes repercussões após um incidente. Além disso, reduzem os custos operacionais, pois é possível pagar apenas pelos recursos usados pela empresa em um modelo baseado em assinatura.

Respostas aos Exercícios Exploratórios

1. Levando em conta o custo e o desempenho, quais as distribuições mais adequadas para uma empresa que tem como objetivo reduzir custos com licenças sem prejudicar o desempenho? Justifique sua resposta.

   Uma das distribuições mais adequadas ao uso empresarial é o CentOS. Ela incorpora todos os produtos do Red Hat, que também são usados no sistema operacional comercial dessa empresa, sem deixar de ser gratuita. Da Mesma maneira, os lançamentos do Ubuntu LTS garantem o suporte por períodos mais longos. As versões estáveis de Debian GNU/Linux também são frequentemente usadas em ambientes empresariais.

2. Quais são as principais vantagens do Raspberry Pi e quais funções ele pode realizar em um contexto empresarial?

   O Raspberry Pi, embora pequeno, funciona como um computador normal. Além disso, custa pouco e pode lidar com tráfego web e muitas outras funcionalidades. Pode ser usado como servidor ou firewall, como placa principal de um robô e muitos outros pequenos dispositivos.

3. Quais distribuições são propostas pelo Amazon Cloud Services e pelo Google Cloud? Cite pelo menos três mais comuns e duas diferentes.

   As distribuições em comum entre o Amazon e o Google Cloud Services são Ubuntu, CentOS e Red Hat Enterprise Linux. Cada um desses provedores de serviços na nuvem propõe distribuições específicas. A Amazon tem o Amazon Linux e o Kali Linux, ao passo que o Google oferece o uso de FreeBSD e Windows Servers.

1.2 Principais Aplicações Open Source

Referência ao LPI objectivo

Linux Essentials version 1.6, Exam 010, Objective 1.2

Peso

2

Áreas chave de conhecimento

- Aplicações Desktop
- Aplicações em Servidores
- Linguagens de desenvolvimento
- Ferramentas de gerenciamento de pacotes e repositórios

Segue uma lista parcial de arquivos, termos e utilitários utilizados

- OpenOffice.org, LibreOffice, Thunderbird, Firefox, GIMP
- Nextcloud, ownCloud
- Apache HTTPD, NGINX, MariaDB, MySQL, NFS, Samba
- C, Java, JavaScript, Perl, shell, Python, PHP
- dpkg, apt-get, rpm, yum

1.2 Lição 1

Certificação:	Linux Essentials
Versão:	1.6
Tópico:	1 A Comunidade Linux e uma carreira em open source
Objetivo:	1.2 Os principais aplicativos de código aberto
Lição:	1 de 1

Introdução

Um aplicativo é um programa de computador cuja função não está diretamente ligada ao funcionamento interno da máquina, mas sim a tarefas realizadas pelo usuário. As distribuições Linux oferecem muitas opções de aplicativos para executar uma variedade de tarefas, como programas de escritório, navegadores web, programas de reprodução e edição de mídia etc. Frequentemente existe mais de um aplicativo ou ferramenta para executar um determinado trabalho. Cabe ao usuário escolher o aplicativo que melhor se adapta às suas necessidades.

Pacotes de software

Quase todas as distribuições Linux vêm com um conjunto pré-instalado de aplicativos por padrão. Além desses aplicativos, as distribuições têm um repositório de pacotes com uma vasta coleção de programas disponíveis para instalação através de seu *gerenciador de pacotes*. Embora as diversas distribuições ofereçam mais ou menos os mesmos aplicativos, há vários sistemas diferentes de gerenciamento de pacotes de acordo com as distribuições. Por exemplo, Debian, Ubuntu e Linux Mint usam as ferramentas `dpkg`, `apt-get` e `apt` para instalar pacotes de software chamados geralmente de *pacotes DEB*. Já distribuições como Red Hat, Fedora e CentOS usam os comandos

`rpm`, `yum` e `dnf`, que por sua vez instalam *pacotes RPM*. Como o empacotamento de aplicativos é diferente para cada família de distribuições, é fundamental instalar pacotes do repositório correto de cada distribuição. O usuário final normalmente não precisa se preocupar com esses detalhes, já que o gerenciador de pacotes da distribuição escolhe os pacotes corretos, as dependências solicitadas e as atualizações futuras. As dependências são os pacotes auxiliares de que os programas necessitam. Por exemplo, se uma biblioteca oferece funções para gerenciar imagens no formato JPEG que são usadas por diversos programas, ela estará muito provavelmente em um pacote próprio, do qual dependerão todos os aplicativos que usarem a biblioteca em questão.

Os comandos `dpkg` e `rpm` operam em arquivos empacotados individuais. Na prática, quase todas as tarefas de gerenciamento de pacotes são executadas pelos comandos `apt-get` ou `apt` em sistemas que usam pacotes DEB, ou por `yum` ou `dnf` em sistemas que usam pacotes RPM. Esses comandos funcionam com catálogos de pacotes, podem baixar novos pacotes e suas dependências e verificar se existem novas versões dos pacotes instalados.

Instalação do pacote

Suponha que você tenha ouvido falar de um comando chamado `figlet`, que mostra um texto ampliado no terminal, e quer experimentá-lo. No entanto, aparece a seguinte mensagem após a execução do comando `figlet`:

```
$ figlet
-bash: figlet: command not found
```

Isso provavelmente indica que o pacote não está instalado em seu sistema. Se sua distribuição funciona com pacotes DEB, você pode procurar os repositórios adequados usando `apt-cache search package_name` ou `apt search package_name`. O comando `apt-cache` é usado para procurar por pacotes e listar informações sobre os pacotes disponíveis. O seguinte comando procura por quaisquer ocorrências do termo “figlet” nos nomes e descrições dos pacotes:

```
$ apt-cache search figlet
figlet - Make large character ASCII banners out of ordinary text
```

A pesquisa identificou um pacote chamado *figlet* que corresponde ao comando ausente. A instalação e a remoção de um pacote requerem permissões especiais concedidas somente ao administrador do sistema: o usuário chamado `root`. Em sistemas desktop, usuários comuns podem instalar ou remover pacotes prefixando o comando `sudo` aos comandos de instalação/remoção. Esse comando exige sua senha para ser executado. Para pacotes DEB, a instalação é realizada com o comando `apt-get install package_name` ou `apt install`

`package_name`:s

```
$ sudo apt-get install figlet
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  figlet
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
```

Nesse momento, o pacote será baixado e instalado no sistema. Quaisquer dependências de que o pacote eventualmente precise também serão baixadas e instaladas:

```
Need to get 184 kB of archives.
After this operation, 741 kB of additional disk space will be used.
Get:1 http://archive.raspbian.org/raspbian stretch/main armhf figlet armhf 2.2.5-2 [184 kB]
Fetched 184 kB in 0s (213 kB/s)
Selecting previously unselected package figlet.
(Reading database ... 115701 files and directories currently installed.)
Preparing to unpack .../figlet_2.2.5-2_armhf.deb ...
Unpacking figlet (2.2.5-2) ...
Setting up figlet (2.2.5-2) ...
update-alternatives: using /usr/bin/figlet-figlet to provide /usr/bin/figlet (figlet) in
auto mode
Processing triggers for man-db (2.7.6.1-2) ...
```

Após o download ser concluído, todos os arquivos são copiados para os locais apropriados, qualquer configuração adicional é executada e o comando se torna disponível:

```
$ figlet Awesome!
    _                                          _
   / \__      _____  ___  ___  _ __ ___   ___| |
  / _ \ \ /\ / / _ \/ __|/ _ \| '_ ` _ \ / _ \ |
 / ___ \ V  V /  __/\__ \ (_) | | | | | |  __/_|
/_/   \_\_/\_/ \___||___/\___/|_| |_| |_|\___(_)
```

Nas distribuições baseadas em pacotes RPM, as pesquisas são feitas usando `yum search package_name` ou `dnf search package_name`. Digamos que você queira exibir um texto de uma forma mais irreverente, seguido por um desenho de vaquinha, mas não sabe bem qual pacote é capaz de executar essa tarefa. Como no caso dos pacotes DEB, os comandos de busca RPM aceitam

termos descritivos:

```
$ yum search speaking cow
Last metadata expiration check: 1:30:49 ago on Tue 23 Apr 2019 11:02:33 PM -03.
=================== Name & Summary Matched: speaking, cow ====================
cowsay.noarch : Configurable speaking/thinking cow
```

Após encontrar um pacote adequado no repositório, ele pode ser instalado com `yum install package_name` ou `dnf install package_name`:

```
$ sudo yum install cowsay
Last metadata expiration check: 2:41:02 ago on Tue 23 Apr 2019 11:02:33 PM -03.
Dependencies resolved.
================================================================================
 Package         Arch           Version               Repository         Size
================================================================================
Installing:
 cowsay          noarch         3.04-10.fc28          fedora             46 k

Transaction Summary
================================================================================
Install  1 Package

Total download size: 46 k
Installed size: 76 k
Is this ok [y/N]: y
```

Mais uma vez, o pacote desejado e todas as suas possíveis dependências serão baixados e instalados:

```
Downloading Packages:
cowsay-3.04-10.fc28.noarch.rpm                  490 kB/s |  46 kB     00:00
--------------------------------------------------------------------------------
Total                                            53 kB/s |  46 kB     00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
Preparing        :                                                        1/1
Installing       : cowsay-3.04-10.fc28.noarch                             1/1
```

```
Running scriptlet: cowsay-3.04-10.fc28.noarch                             1/1
Verifying        : cowsay-3.04-10.fc28.noarch                             1/1

Installed:
cowsay.noarch 3.04-10.fc28

Complete!
```

O comando `cowsay` faz exatamente o que seu nome implica:

```
$ cowsay "Brought to you by yum"
 _________________________
< Brought to you by yum >
 -------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Embora possam parecer inúteis, os comandos `figlet` e `cowsay` oferecem uma maneira de chamar a atenção de outros usuários para informações relevantes.

Remoção de Pacotes

Os mesmos comandos usados para instalar pacotes são usados para removê-los. Todos os comandos aceitam a palavra-chave `remove` para desinstalar um pacote instalado: `apt-get remove package_name` ou `apt remove package_name` para pacotes DEB e `yum remove package_name` ou `dnf remove package_name` para pacotes RPM. O comando `sudo` também é necessário para executar a remoção. Por exemplo, para remover o pacote *figlet* previamente instalado em uma distribuição baseada em DEB:

```
$ sudo apt-get remove figlet
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages will be REMOVED:
  figlet
0 upgraded, 0 newly installed, 1 to remove and 0 not upgraded.
After this operation, 741 kB disk space will be freed.
```

```
Do you want to continue? [Y/n] Y
```

Depois de confirmar a operação, o pacote é removido do sistema:

```
(Reading database ... 115775 files and directories currently installed.)
Removing figlet (2.2.5-2) ...
Processing triggers for man-db (2.7.6.1-2) ...
```

Nos sistemas baseados em RPM, o procedimento é semelhante. Por exemplo, para remover o pacote *cowsay* instalado anteriormente em uma distribuição baseada em RPM:

```
$ sudo yum remove cowsay
Dependencies resolved.
================================================================================
 Package          Arch             Version                Repository         Size
================================================================================
Removing:
 cowsay           noarch           3.04-10.fc28           @fedora            76 k

Transaction Summary
================================================================================
Remove  1 Package

Freed space: 76 k
Is this ok [y/N]: y
```

Da mesma forma, uma confirmação é solicitada e o pacote é apagado do sistema:

```
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                        1/1
  Erasing          : cowsay-3.04-10.fc28.noarch                             1/1
  Running scriptlet: cowsay-3.04-10.fc28.noarch                             1/1
  Verifying        : cowsay-3.04-10.fc28.noarch                             1/1

Removed:
  cowsay.noarch 3.04-10.fc28
```

```
Complete!
```

Os arquivos de configuração dos pacotes removidos são mantidos no sistema e podem ser utilizados novamente se o pacote for reinstalado no futuro.

Aplicativos de escritório

Os aplicativos de escritório são utilizados para editar arquivos de texto, apresentações, planilhas e outros formatos comumente utilizados num ambiente de escritório. Esses aplicativos são normalmente organizados em coleções chamadas *office suites*, ou suíte de escritório.

Durante muito tempo, a suíte de escritório mais usada no Linux foi o *OpenOffice.org*. O OpenOffice.org era uma versão open source da *StarOffice suite*, produzida pela *Sun Microsystems*. Alguns anos depois, a Sun foi adquirida pela *Oracle Corporation*, que por sua vez transferiu o projeto para a *Apache Foundation*, e o OpenOffice.org foi rebatizado como *Apache OpenOffice*. Nesse meio tempo, outra suíte de escritório baseada no mesmo código fonte foi lançada pela *Document Foundation*, que a batizou de *LibreOffice*.

Os dois projetos têm as mesmas características básicas e são compatíveis com os formatos de documentos do *Microsoft Office*. No entanto, o formato de documento preferido é o *Open Document Format*, um formato de arquivo totalmente aberto e padrão ISO. A utilização de arquivos ODF garante que os documentos possam ser transferidos entre sistemas operacionais e aplicativos de diferentes fornecedores, como o Microsoft Office. Os principais aplicativos oferecidos pelo OpenOffice/LibreOffice são:

Writer
: Editor de texto

Calc
: Planilhas

Impress
: Apresentações

Draw
: Desenho vetorial

Math
: Fórmulas matemáticas

Base

Banco de dados

Tanto o LibreOffice quanto o Apache OpenOffice são softwares de código aberto, mas o LibreOffice é licenciado sob a LGPLv3 e o Apache OpenOffice traz a licença Apache License 2.0. A distinção de licenciamento implica que o LibreOffice pode incorporar melhorias feitas pelo Apache OpenOffice, mas o Apache OpenOffice não pode incorporar melhorias feitas pelo LibreOffice. Essa, junto com uma comunidade mais ativa de desenvolvedores, é a razão pela qual a maioria das distribuições adota o LibreOffice como seu conjunto padrão de aplicativos de escritório.

Navegadores web

Para a maioria dos usuários, o principal objetivo de um computador é fornecer acesso à Internet. Hoje em dia, as páginas web podem funcionar como um aplicativo completo, com a vantagem de serem acessíveis de qualquer lugar, sem a necessidade de instalar software extra. Isso faz do navegador web o aplicativo mais importante do sistema operacional, pelo menos para o usuário médio.

TIP

Uma das melhores fontes para aprender mais sobre desenvolvimento web é o MDN Web Docs, disponível em https://developer.mozilla.org/. Mantido pela Mozilla, o site está repleto de tutoriais para iniciantes e materiais de referência sobre as mais modernas tecnologias web.

Os principais navegadores web no ambiente Linux são o *Google Chrome* e o *Mozilla Firefox*. O Chrome é um navegador web mantido pelo Google, mas baseado no navegador de código aberto *Chromium*, que pode ser instalado através do gerenciador de pacotes da distribuição e é totalmente compatível com o Chrome. Mantido pela Mozilla, uma organização sem fins lucrativos, o Firefox é um navegador cujas origens estão ligadas ao Netscape, o primeiro navegador web popular a adotar o modelo open source. A Fundação Mozilla está profundamente envolvida com o desenvolvimento dos padrões abertos subjacentes à web moderna.

A Mozilla também desenvolve outros aplicativos, como o cliente de email *Thunderbird*. Muitos usuários optam por usar webmail em vez de um programa de email dedicado, mas um cliente como o Thunderbird oferece funcionalidades extras e integra-se melhor com outras aplicações no desktop.

Multimídia

Comparado com os aplicativos web disponíveis, os aplicativos para desktop ainda são a melhor opção para a criação de conteúdo multimídia. As atividades relacionadas a multimídia, como a

renderização de vídeo, frequentemente exigem quantidades elevadas de recursos do sistema, que podem ser mais bem administrados por um aplicativo desktop local. Listamos abaixo alguns dos aplicativos multimídia mais populares do ambiente Linux e seus usos.

Blender

Um renderizador 3D para criar animações. O Blender também pode ser usado para exportar objetos 3D para serem impressos em uma impressora 3D.

GIMP

Um editor de imagens completo, que pode ser comparado com o *Adobe Photoshop*, mas com seus próprios conceitos e ferramentas para trabalhar com imagens. O GIMP pode ser usado para criar, editar e salvar a maioria dos arquivos bitmap, como JPEG, PNG, GIF, TIFF e muitos outros.

Inkscape

Um editor de gráficos vetoriais, semelhante ao *Corel Draw* ou ao *Adobe Illustrator*. O formato padrão do Inkscape é SVG, um padrão aberto para gráficos vetoriais. Os arquivos SVG podem ser abertos por qualquer navegador web e, devido à sua natureza de gráfico vetorial, podem ser usados em layouts flexíveis de páginas web.

Audacity

Um editor de áudio. O Audacity pode ser usado para filtrar, aplicar efeitos e converter entre diferentes formatos de áudio, como MP3, WAV, OGG, FLAC etc.

ImageMagick

O ImageMagick é uma ferramenta de linha de comando usada para converter e editar a maioria dos tipos de arquivos de imagem. Também pode ser usado para criar documentos PDF a partir de arquivos de imagem e vice-versa.

Também existem muitos aplicativos dedicados à reprodução de mídia. O programa mais popular para a reprodução de vídeo é o *VLC*, mas alguns usuários preferem outras alternativas, como o *smplayer*. A reprodução de música local também traz muitas opções, como o *Audacious*, o *Banshee* e o *Amarok*, que também podem gerenciar uma coleção local de arquivos de áudio.

Programas de servidor

Quando um navegador web carrega uma página de um site, na verdade ele se conecta a um computador remoto e pede uma informação específica. Nesse cenário, o computador que executa o navegador web é chamado de *cliente*, e o computador remoto, *servidor*.

O computador servidor, que pode ser um computador desktop comum ou hardware especializado,

necessita de um programa específico para gerenciar cada tipo de informação que irá fornecer. No que tange ao envio de páginas web, a maioria dos servidores ao redor do mundo lança mão de programas de servidor de código aberto. Esse programa de servidor em particular é chamado de *HTTP server* (HTTP significa *Hyper Text Transfer Protocol*, ou Protocolo de Transferência de Hipertexto), e os mais populares são *Apache*, *Nginx* e *lighttpd*.

Mesmo as páginas web mais simples podem ter muitas solicitações, que vão desde arquivos comuns — o que se chama conteúdo estático — até o conteúdo dinâmico renderizado a partir de diversas fontes. O papel de um servidor HTTP é coletar e enviar todos os dados solicitados de volta para o navegador, que então organiza o conteúdo conforme definido pelo documento HTML recebido (HTML significa *Hyper Text Markup Language*, ou Linguagem de Marcação de Hipertexto) e outros arquivos anexos. Portanto, a renderização de uma página web envolve operações executadas no lado do servidor e operações executadas no lado do cliente. Ambos os lados podem usar scripts personalizados para realizar tarefas específicas. No lado do servidor HTTP, a linguagem de scripts PHP é comumente usada. Já no lado do cliente usa-se o JavaScript.

Os programas de servidor podem fornecer todo tipo de informação. Não é incomum que um programa de servidor solicite informações fornecidas por outros programas servidores. Esse é o caso quando um servidor HTTP pede informações fornecidas por um servidor de banco de dados.

Por exemplo, quando uma página dinâmica é solicitada, o servidor HTTP normalmente consulta um banco de dados para coletar todas as informações necessárias e envia o conteúdo dinâmico de volta para o cliente. De maneira semelhante, quando um usuário se inscreve em um site, o servidor HTTP reúne os dados enviados pelo cliente e os armazena em um banco de dados.

Um banco de dados é um conjunto organizado de informações. Um servidor de banco de dados armazena conteúdos de maneira formatada, tornando possível ler, escrever e vincular grandes quantidades de dados com alta velocidade. Os servidores de banco de dados de código aberto são usados em muitas aplicações, não apenas na Internet. Mesmo os aplicativos locais podem armazenar dados, conectando-se a um servidor de banco de dados local. O tipo mais comum de banco de dados é o *banco de dados relacional*, no qual os dados são organizados em tabelas predefinidas. Os bancos de dados relacionais de código aberto mais populares são *MariaDB* (derivado do *MySQL*) e *PostgreSQL*.

Compartilhamento de dados

Em redes locais, como as encontradas em escritórios e casas, é desejável que os computadores não somente possam acessar a Internet, como também sejam capazes de se comunicar uns com os outros. Às vezes um computador age como um servidor, outras vezes o mesmo computador age como um cliente, o que é necessário quando se quer acessar arquivos em outro computador na rede — por exemplo, acessar um arquivo armazenado em um computador desktop a partir de um

dispositivo portátil — sem o incômodo de copiá-lo para um drive USB ou algo do tipo.

Entre máquinas com Linux, usa-se frequentemente o *NFS* (*Network File System*, ou Sistema de Arquivos de Rede). O protocolo NFS é a forma padrão de compartilhar sistemas de arquivos em redes equipadas apenas com máquinas Unix/Linux. Com o NFS, um computador pode compartilhar um ou mais diretórios com computadores específicos na rede, permitindo que eles leiam e escrevam arquivos nesses diretórios. O NFS pode ser usado inclusive para compartilhar a árvore de diretórios inteira de um sistema operacional com clientes que podem assim inicializar a partir dela. Esses computadores, chamados *thin clients* ou "clientes magros", são geralmente usados em grandes redes para evitar a manutenção de cada sistema operacional individual em uma rede.

Se existem outros tipos de sistemas operacionais ligados à rede, recomenda-se a utilização de um protocolo de compartilhamento de dados que possa ser compreendido por todos eles. O *Samba* cumpre esse requisito. O Samba implementa um protocolo de compartilhamento de arquivos na rede originalmente pensado para o sistema operacional Windows, mas hoje é compatível com todos os principais sistemas operacionais. Com o Samba, os computadores na rede local não só podem compartilhar arquivos, como também impressoras.

Em algumas redes locais, a autorização dada pelo login em uma estação de trabalho é concedida por um servidor central, chamado *controlador de domínio*, que gerencia o acesso a vários recursos locais e remotos. O controlador de domínio é um serviço fornecido pelo *Active Directory* da Microsoft. As estações de trabalho Linux podem ser associadas a um controlador de domínio usando Samba ou um subsistema de autenticação chamado *SSSD*. A partir da versão 4, o Samba também pode funcionar como controlador de domínio em redes heterogêneas.

Se o objetivo é implementar uma solução de computação em nuvem capaz de fornecer diversos métodos de compartilhamento de dados baseados na web, duas alternativas devem ser consideradas: *ownCloud* e *Nextcloud*. Os dois projetos são muito semelhantes porque o Nextcloud é um derivado do ownCloud, o que não é incomum entre projetos open source. Esses derivados são geralmente chamados de *fork* (bifurcação ou ramificação). Ambos oferecem os mesmos recursos básicos: compartilhamento e sincronização de arquivos, espaços de trabalho colaborativos, calendário, contatos e email, tudo através de interfaces para desktop, celular e web. O Nextcloud oferece também conferência em áudio e vídeo, ao passo que o ownCloud concentra-se mais em compartilhamento de arquivos e integração com software de terceiros. Há muito mais recursos disponíveis na forma de plugins que podem ser ativados no momento em que forem necessários.

Tanto o ownCloud quanto o Nextcloud oferecem uma versão paga com recursos extras e suporte estendido. O que os torna diferentes de outras soluções comerciais é a possibilidade de instalar o NextCloud ou o ownCloud em um servidor privado, gratuitamente, evitando assim que se

mantenham dados confidenciais em um servidor desconhecido. Como todos os serviços dependem de comunicação HTTP e são escritos em PHP, a instalação deve ser realizada em um servidor web previamente configurado, como o Apache. Se você estiver pensando em instalar o ownCloud ou o Nextcloud em seu próprio servidor, não se esqueça de habilitar também o HTTPS para criptografar todas as conexões à sua nuvem.

Administração de redes

A comunicação entre computadores só é possível se a rede estiver funcionando corretamente. Normalmente, a configuração da rede é feita por um conjunto de programas executados no roteador, responsáveis por configurar e verificar a disponibilidade da rede. Para isso, dois serviços de rede básicos são usados: *DHCP* (*Dynamic Host Configuration Protocol*, ou Protocolo de Configuração Dinâmica de Host) e *DNS* (*Domain Name System*, ou Sistema de Nomes de Domínio).

O DHCP é responsável por atribuir um endereço IP ao host quando um cabo de rede está conectado ou quando o dispositivo entra em uma rede sem fio. Quando nos conectamos à Internet, o servidor DHCP do provedor de acesso fornece um endereço IP ao dispositivo solicitante. Um servidor DHCP é muito útil também em redes locais, para fornecer endereços IP automaticamente a todos os dispositivos conectados. Se o DHCP não estiver configurado ou se não estiver funcionando corretamente, será necessário configurar manualmente o endereço IP de cada dispositivo conectado à rede. Não é prático definir manualmente os endereços IP em redes grandes ou mesmo em redes pequenas, e por essa razão a maioria dos roteadores de rede vem com um servidor DHCP pré-configurado por padrão.

O endereço IP é necessário para possibilitar a comunicação com um outro dispositivo em uma rede IP, mas nomes de domínio como `www.lpi.org` são muito mais fáceis de lembrar do que um número IP como `203.0.0.113.165`. O nome de domínio sozinho, entretanto, não basta para estabelecer a comunicação através da rede. Assim, o nome de domínio precisa ser traduzido em forma de endereço IP por um servidor DNS. O endereço IP do servidor DNS é fornecido pelo servidor DHCP do provedor de acesso e é usado por todos os sistemas conectados para traduzir nomes de domínio como endereços IP.

As configurações do DHCP e do DNS podem ser modificadas através da interface web do roteador. Por exemplo, é possível restringir a atribuição de IP apenas a dispositivos conhecidos ou associar um endereço IP fixo a máquinas específicas. Também é possível alterar o servidor DNS padrão fornecido pelo provedor de acesso. Alguns servidores DNS de terceiros, como os fornecidos pelo Google ou OpenDNS, podem ter um tempo de resposta mais rápido e funcionalidades adicionais.

Linguagens de programação

Todos os programas de computador (programas de cliente e servidor, aplicativos para desktop e o próprio sistema operacional) são feitos a partir de uma ou mais linguagens de programação. Os programas podem ser um único arquivo ou um sistema complexo de centenas de arquivos, que o sistema operacional trata como uma seqüência de instruções a ser interpretada e executada pelo processador e outros dispositivos.

Existem inúmeras linguagens de programação para diferentes finalidades, e os sistemas Linux fornecem muitas delas. Como o software de código aberto também inclui as fontes dos programas, os sistemas Linux oferecem aos desenvolvedores condições perfeitas para entender, modificar ou criar software de acordo com suas próprias necessidades.

Cada programa começa como um arquivo de texto, chamado *código fonte*. Esse código fonte é escrito em uma linguagem mais ou menos amigável que descreve o que o programa está fazendo. Um processador de computador não pode executar diretamente esse código. Assim, nas *linguagens compiladas*, o código fonte é convertido em um *arquivo binário*, que pode por sua vez ser executado pelo computador. Um programa chamado *compilador* é responsável por fazer a conversão do código fonte para a forma executável. Como o arquivo binário é compilado para um modelo específico de processador, o programa pode ter de ser recompilado para rodar em outro tipo de computador.

Nas *linguagens interpretadas*, o programa não precisa ser compilado previamente. Em vez disso, um *interpretador* lê o código fonte e executa suas instruções toda vez que o programa é executado. Isso torna o desenvolvimento mais fácil e rápido, mas, ao mesmo tempo, os programas interpretados tendem a ser mais lentos do que os programas compilados.

Eis algumas das linguagens de programação mais populares:

JavaScript

O JavaScript é uma linguagem de programação usada principalmente em páginas web. No início, os aplicativos em JavaScript eram muito simples, como rotinas de validação de formulários. Hoje em dia, o JavaScript é considerado uma linguagem de primeira classe e é usado para criar aplicações muito complexas não só na web, mas também em servidores e dispositivos móveis.

C

A linguagem de programação C está intimamente relacionada com os sistemas operacionais, particularmente o Unix, mas é usada para escrever qualquer tipo de programa para quase qualquer tipo de dispositivo. As grandes vantagens do C são a flexibilidade e a velocidade. O mesmo código fonte escrito em C pode ser compilado para rodar em diferentes plataformas e

sistemas operacionais, com pouca ou nenhuma modificação. Após ser compilado, porém, o programa só roda no sistema alvo.

Java

O aspecto principal do Java é que os programas escritos nessa linguagem são portáteis, o que significa que o mesmo programa pode ser executado em diferentes sistemas operacionais. Apesar do nome, o Java não tem relação com o JavaScript.

Perl

O Perl é uma linguagem de programação mais usada para processar conteúdo de texto. Tem uma forte ênfase em expressões regulares, o que faz do Perl uma linguagem adequada para filtragem e análise de texto.

Shell

O shell, particularmente o Bash shell, não é apenas uma linguagem de programação, mas uma interface interativa para executar outros programas. Os programas em Shell, conhecidos como *shell scripts*, podem automatizar tarefas complexas ou repetitivas no ambiente de linha de comando.

Python

O Python é uma linguagem de programação muito popular entre estudantes e profissionais não diretamente envolvidos com ciência da computação. Apesar de ter recursos avançados, o Python é uma boa maneira de começar a aprender programação devido à sua facilidade de uso.

PHP

O PHP é mais usado como uma linguagem de script do lado do servidor para gerar conteúdo para a web. A maioria das páginas HTML online não são arquivos estáticos, mas conteúdo dinâmico gerado pelo servidor a partir de várias fontes, como bancos de dados. Os programas em PHP — às vezes chamados apenas de páginas PHP ou scripts PHP — são freqüentemente usados para gerar esse tipo de conteúdo. O termo LAMP vem da combinação de um sistema operacional Linux, um servidor Apache HTTP, um banco de dados MySQL (ou MariaDB) e programação em PHP. Os servidores LAMP são uma solução muito popular para rodar servidores web. Além do PHP, todas as linguagens de programação descritas anteriormente podem ser usadas para implementar esse tipo de aplicação.

C e Java são linguagens compiladas. A fim de ser executado pelo sistema, o código fonte escrito em C é convertido em código de máquina binário, enquanto o código fonte em Java é convertido em *bytecode* executado em um ambiente de software especial chamado *Java Virtual Machine*. JavaScript, Perl, Shell script, Python e PHP são linguagens interpretadas, também chamadas de

linguagens de script.

Exercícios Guiados

1. Para cada um dos seguintes comandos, identifique se ele está associado ao *sistema de empacotamento Debian* ou ao *sistema de empacotamento Red Hat*:

`dpkg`	
`rpm`	
`apt-get`	
`yum`	
`dnf`	

2. Qual comando poderia ser usado para instalar o Blender no Ubuntu? Após a instalação, como o programa pode ser executado?

3. Qual aplicativo da suíte LibreOffice pode ser usado para trabalhar com planilhas eletrônicas?

4. Qual navegador web de código aberto foi usado como base para o desenvolvimento do Google Chrome?

5. O SVG é um padrão aberto para gráficos vetoriais. Qual é o aplicativo mais popular para editar arquivos SVG em sistemas Linux?

6. Para cada um dos seguintes formatos de arquivo, escreva o nome de um aplicativo capaz de abrir e editar o arquivo correspondente:

`png`	
`doc`	
`xls`	
`ppt`	
`wav`	

7. Que pacote de software permite o compartilhamento de arquivos entre máquinas Linux e Windows através da rede local?

Exercícios Exploratórios

1. Você sabe que os arquivos de configuração são preservados mesmo que o pacote associado seja removido do sistema. Como seria possível remover automaticamente o pacote chamado *cups* e seus arquivos de configuração de um sistema baseado em DEB?

2. Suponha que você tem muitos arquivos de imagem TIFF e quer convertê-los para JPEG. Qual pacote de software poderia ser usado para converter esses arquivos diretamente na linha de comando?

3. Qual pacote de software você precisa instalar para poder abrir documentos do Microsoft Word enviados a você por um usuário do Windows?

4. Todos os anos, o site linuxquestions.org promove uma pesquisa sobre os aplicativos Linux mais populares. Visite https://www.linuxquestions.org/questions/2018-linuxquestions-org-members-choice-awards-128/ e descubra quais são os aplicativos de desktop mais populares entre os utilizadores experientes de Linux.

Sumário

Nesta lição, você aprendeu:

- Os sistemas de gerenciamento de pacotes usados nas principais distribuições Linux
- Aplicativos de código aberto capazes de editar formatos de arquivos populares
- Os programas de servidor subjacentes a muitos serviços importantes de Internet e rede local
- Linguagens de programação comuns e seus usos

Respostas aos Exercícios Guiados

1. Para cada um dos seguintes comandos, identifique se ele está associado ao sistema de empacotamento _Debian ou ao sistema de empacotamento _Red Hat:

`dpkg`	sistema de empacotamento Debian
`rpm`	sistema de empacotamento Red Hat
`apt-get`	sistema de empacotamento Debian
`yum`	sistema de empacotamento Red Hat
`dnf`	sistema de empacotamento Red Hat

2. Qual comando poderia ser usado para instalar o Blender no Ubuntu? Após a instalação, como o programa pode ser executado?

   O comando `apt-get install blender`. O nome do pacote deve ser especificado em minúsculas. O programa pode ser executado diretamente do terminal com o comando `blender` ou escolhendo-o no menu de aplicativos.

3. Qual aplicativo da suíte LibreOffice pode ser usado para trabalhar com planilhas eletrônicas?

   Calc

4. Qual navegador web de código aberto foi usado como base para o desenvolvimento do Google Chrome?

   Chromium

5. O SVG é um padrão aberto para gráficos vetoriais. Qual é o aplicativo mais popular para editar arquivos SVG em sistemas Linux?

   Inkscape

6. Para cada um dos seguintes formatos de arquivo, escreva o nome de um aplicativo capaz de abrir e editar o arquivo correspondente:

`png`	Gimp
`doc`	LibreOffice Writer
`xls`	LibreOffice Calc
`ppt`	LibreOffice Impress

`wav`	Audacity

7. Que pacote de software permite o compartilhamento de arquivos entre máquinas Linux e Windows através da rede local?

   Samba

Respostas aos Exercícios Exploratórios

1. Você sabe que os arquivos de configuração são preservados mesmo que o pacote associado seja removido do sistema. Como seria possível remover automaticamente o pacote chamado *cups* e seus arquivos de configuração de um sistema baseado em DEB?

   `apt-get purge cups`

2. Suponha que você tem muitos arquivos de imagem TIFF e quer convertê-los para JPEG. Qual pacote de software poderia ser usado para converter esses arquivos diretamente na linha de comando?

   ImageMagick

3. Qual pacote de software você precisa instalar para poder abrir documentos do Microsoft Word enviados a você por um usuário do Windows?

   LibreOffice ou OpenOffice

4. Todos os anos, o site linuxquestions.org promove uma pesquisa sobre os aplicativos Linux mais populares. Visite https://www.linuxquestions.org/questions/2018-linuxquestions-org-members-choice-awards-128/ e descubra quais são os aplicativos de desktop mais populares entre os utilizadores experientes de Linux.

   Navegador: Firefox. Cliente de email: Thunderbird. Reprodução de mídia: VLC. Editor de imagens rasterizadas: GIMP.

1.3 Entendendo o Software Open Source e suas Licenças

Referência ao LPI objectivo

Linux Essentials version 1.6, Exam 010, Objective 1.3

Peso

1

Áreas chave de conhecimento

- Filosofia do Código Aberto
- Licenciamento
- Free Software Foundation (FSF), Open Source Initiative (OSI)

Segue uma lista parcial de arquivos, termos e utilitários utilizados

- GPL, BSD, Creative Commons, Copyleft, Permissive
- Free Software, Open Source Software, FOSS, FLOSS
- Modelos de negócios Open Source

1.3 Lição 1

Certificação:	Linux Essentials
Versão:	1.6
Tópico:	1 A Comunidade Linux e uma carreira em open source
Objetivo:	1.3 Software open source e licenciamento
Lição:	1 de 1

Introdução

Embora os termos *software livre* e *software de código aberto* sejam amplamente utilizados, ainda existem alguns equívocos sobre seu significado. Em particular, o conceito de “liberdade” precisa ser examinado mais de perto. Vamos começar com a definição dos dois termos.

Definição de Software Livre e de Código Aberto

Critérios do Software Livre

Antes de mais nada, “livre”, no contexto de software livre, não tem nada a ver com “gratuito”, ou, como resume o fundador da *Free Software Foundation* (FSF), Richard Stallman:

> Para entender o conceito, você deve pensar em “livre” como em “livre expressão”, não como em “cerveja grátis”.
>
> — Richard Stallman, What is free software? (O que é software livre?)

Independentemente de você ter de pagar ou não pelo software, existem quatro critérios que definem um software livre. Richard Stallman descreve estes critérios como “as quatro liberdades essenciais”, cuja contagem começa em zero:

- “A liberdade de executar o programa como você deseja, para qualquer finalidade (liberdade 0).”

  Onde, como e para qual propósito o software é usado não é algo que possa ser estipulado e nem restringido.

- “A liberdade de estudar como o programa funciona e de alterá-lo para que ele compute seus dados da maneira como você deseja (liberdade 1). O acesso ao código fonte é um pré-requisito para isso.”

  Qualquer pessoa pode alterar o software de acordo com suas idéias e necessidades, o que por sua vez pressupõe que o chamado *código fonte*, ou seja, todos os arquivos que compõem um software, devem estar disponíveis em um formato legível pelos programadores. E, é claro, esse direito se aplica a um único usuário que pode querer adicionar um único recurso, mas também a empresas de software que constroem sistemas complexos, como sistemas operacionais de smartphones ou firmware de roteadores.

- “A liberdade de redistribuir cópias para que você possa ajudar os outros (liberdade 2).”

  Esta liberdade encoraja explicitamente cada usuário a compartilhar o software com outras pessoas. Trata-se, portanto, da mais ampla distribuição possível e, portanto, da maior comunidade possível de usuários e desenvolvedores que, com base nestas liberdades, desenvolvem e aprimoram ainda mais o software para o benefício de todos.

- “A liberdade de distribuir cópias de suas versões modificadas para outras pessoas (liberdade 3). Ao fazer isso, você está dando a toda a comunidade a chance de se beneficiar de suas alterações. O acesso ao código fonte é um pré-requisito para isso.”

  Não se trata apenas da distribuição de software livre, mas da distribuição de software livre *modificado*. Qualquer pessoa que fizer alterações em um software livre tem o direito de disponibilizar as alterações para outros. Se o fizer, é obrigada a fazê-lo também livremente, ou seja, não deve restringir as liberdades originais ao distribuir o software, mesmo que ele tenha sido modificado ou ampliado. Por exemplo, se um grupo de desenvolvedores tiver ideias diferentes dos criadores originais sobre a direção que um software específico deve tomar, podem criar sua própria ramificação de desenvolvimento (chamada também de *bifurcação*) e continuar a desenvolvê-lo como um novo projeto. Mas, obviamente, todas as obrigações associadas a essa liberdade continuam vigentes.

A ênfase na idéia de liberdade também é consistente com o fato de que todo movimento libertário é dirigido *contra* algo; no caso, um oponente que refuta as liberdades postuladas, considera o software como propriedade e deseja mantê-lo guardado a sete chaves. Em contraste com o software livre, esse software é chamado *proprietário*.

Software de Código Aberto versus Software Livre

Para muitos, *software livre* e *software de código aberto* são sinônimos. A abreviação freqüentemente usada *FOSS*, para *Software Livre e de Código Aberto* (em inglês, Free and Open Source Software), enfatiza essa aproximação. FLOSS para *Software Livre/Libre e de Código Aberto* (Free/Libre and Open Source Software) é outro termo popular, que enfatiza inequivocamente a idéia de liberdade também para outros idiomas que não o inglês. No entanto, se considerarmos a origem e o desenvolvimento de ambos os termos, vale a pena diferenciá-los.

O termo *software livre* com a definição das quatro liberdades descritas remonta a Richard Stallman e ao projeto GNU fundado por ele em 1985 — quase 10 anos antes do surgimento do Linux. O nome “GNU não é Unix” descreve a intenção com um toque de humor: o GNU começou como uma iniciativa para desenvolver uma solução tecnicamente convincente — no caso, o sistema operacional Unix — do zero, torná-la disponível para o público em geral e melhorá-la continuamente junto ao público em geral. O fato de o código fonte ser aberto representava simplesmente uma necessidade técnica e organizacional para atingir esse objetivo, mas em sua auto-imagem o movimento do software livre ainda é um movimento *social* e *político* - e, segundo certas fontes, também ideológico.

Com o sucesso do Linux, as possibilidades colaborativas da Internet e os milhares de projetos e empresas que surgiram nesse novo cosmos do software, o aspecto social cada vez mais recuou para o segundo plano. A abertura do código fonte deixou de ser um requisito técnico para se tornar uma característica determinante: quando o código fonte era visível, o software era considerado “de código aberto”. Os motivos sociais deram lugar a uma abordagem mais pragmática no desenvolvimento de software.

O software livre e o software de código aberto trabalham na mesma coisa, com os mesmos métodos e em uma comunidade global de indivíduos, projetos e empresas. Mas por serem oriundos de diferentes direções — uma social e uma pragmática-técnica — às vezes ocorrem conflitos. Eles surgem quando os resultados do trabalho conjunto não correspondem aos objetivos originais de ambos os movimentos. Isso ocorre sobretudo quando o software abre suas fontes, mas não respeita as quatro liberdades do software livre ao mesmo tempo, por exemplo quando há restrições à divulgação, à alteração ou às conexões com outros componentes do software.

A *licença* sob a qual o software é disponibilizado determina a condições a que um software está sujeito em relação ao seu uso, distribuição e modificação. E como os requisitos e motivos podem

ser muito diferentes, inúmeras licenças diferentes foram criadas na área de FOSS. Devido à abordagem muito mais fundamental do movimento do software livre, não é de surpreender que ele não reconheça muitas licenças de código aberto como "livres" e, assim, as rejeite. Inversamente, esse raramente é o caso na abordagem muito mais pragmática do código aberto.

Analisamos abaixo, em poucas palavras, o intricadíssimo campo das licenças.

Licenças

Ao contrário de uma geladeira ou um carro, o software não é um produto *físico*, mas *digital*. Assim, uma empresa não pode de fato transferir a propriedade desse produto vendendo-o e transferindo a posse física — em vez disso, ela transfere os direitos de uso daquele produto e o usuário concorda contratualmente com esses direitos de uso. A definição do que são esses direitos de uso e, principalmente, do que eles *não* são é registrado na licença do software e, assim, é fácil entender a importância das regulações ali contidas.

Ao passo que os grandes fornecedores de software proprietário, como a Microsoft ou a SAP, têm suas próprias licenças feitas sob medida para seus produtos, os defensores do software livre e de código aberto desde o início buscaram a clareza e a universalidade em suas licenças, já que, afinal, todo usuário deve poder compreendê-las e, se necessário, usá-las ele próprio para seus próprios desenvolvimentos.

No entanto, não se deve esconder que esse ideal de simplicidade dificilmente pode ser alcançado, pois é preciso enfrentar demasiados requisitos específicos e entendimentos legais nem sempre compatíveis internacionalmente. Para dar apenas um exemplo: a legislação alemã e a americana são fundamentalmente diferentes em matéria de direitos autorais. De acordo com a legislação alemã, existe uma *pessoa* como *autor* (mais precisamente: *Urheber*), cuja obra é sua *propriedade intelectual*. Embora o autor possa autorizar a utilização da sua obra, ele não pode atribuir ou renunciar à sua autoria. Esta última especificidade não existe na legislação americana. Também aqui existe um autor (que, porém, também pode ser uma empresa ou instituição), mas ele só possui os direitos de exploração, os quais podem ser transferidos totalmente ou em parte, possibilitando que o autor se dissocie completamente de sua obra. Uma licença válida internacionalmente precisa ser interpretada levando-se em conta as diferentes legislações.

Como consequência, existem muitas licenças FOSS que, frequentemente, são bem diferentes. Pior ainda são as diferentes versões de uma mesma licença, ou ainda uma mistura de licenças (dentro de um projeto, ou até ao se conectar vários projetos), que podem causar confusão ou mesmo disputas legais.

Tanto os representantes do software livre quanto os defensores do movimento de código aberto (que tem uma orientação predominantemente econômica) criaram suas próprias organizações,

que hoje são decisivamente responsáveis pela formulação das licenças de software de acordo com seus princípios e apoiam seus membros em sua aplicação.

Copyleft

A já mencionada *Free Software Foundation* (FSF) formulou a *GNU General Public License* (GPL), uma das mais importantes licenças de software livre, usada por muitos projetos, como o kernel do Linux. Além disso, ela lançou licenças que podem ser personalizadas para um determinado projeto, como a *GNU Lesser General Public License* (LGPL), que rege a combinação de software livre com modificações cujo código-fonte não precisa ser divulgada ao público, a *GNU Affero General Public License* (AGPL), que cobre a venda de acesso a software hospedado, ou a *GNU Free Documentation License* (FDL), que estende os princípios da liberdade à documentação do software. Além disso, a FSF faz recomendações pró ou contra licenças de terceiros, e projetos afiliados como o GPL-Violations.org investigam suspeitas de violação de licenças livres.

A FSF chama o princípio segundo o qual uma licença livre também se aplica a variantes modificadas do software de *copyleft* — em contraste com o princípio de direitos autorais restritivos que ela rejeita. A idéia, portanto, é transferir os princípios generosos de uma licença de software da forma mais irrestrita possível para futuras variantes do software, a fim de evitar restrições subseqüentes.

O que parece óbvio e simples, no entanto, leva a complicações consideráveis na prática, razão pela qual os críticos muitas vezes chamam o princípio copyleft de “viral”, já que ele é transmitido para as versões posteriores.

Do que foi dito, pode-se depreender, por exemplo, que dois componentes de software licenciados sob diferentes licenças copyleft podem não ser combináveis entre si, uma vez que ambas as licenças não podem ser transferidas para o produto subsequente ao mesmo tempo. Isso pode se aplicar até mesmo a versões diferentes da mesma licença!

Por essa razão, as licenças ou versões de licenças mais recentes já não abraçam o copyleft com tanto rigor. A já mencionada *GNU Lesser General Public License* (LGPL) é, nesse sentido, uma concessão que permite conectar software livre com componentes “não livres”, como ocorre freqüentemente com as chamadas *bibliotecas*. As bibliotecas contêm subrotinas ou rotinas, que por sua vez são usadas por vários outros programas. Isso leva à situação comum na qual um software proprietário faz apelo à subrotina de uma biblioteca livre.

Outro modo de evitar conflitos de licença é a *licença dual*, na qual um software é licenciado sob diferentes licenças, por exemplo uma licença livre e uma licença proprietária. Um caso típico de uso é uma versão livre de um software que só pode ser usado quando se respeitam as restrições de copyleft e a alternativa de se adquirir o software sob uma licença diferente, que libera o

licenciado de certas restrições em troca de uma taxa que pode ser usada para financiar o desenvolvimento do software.

Portanto, deve ficar claro que a escolha da licença para projetos de software deve ser feita com muita cautela, uma vez que a cooperação com outros projetos, a possibilidade de combinação com outros componentes e também o design futuro do próprio produto dependem disso. O copyleft apresenta desafios especiais aos desenvolvedores nesse aspecto.

Definição de Código Aberto e licenças permissivas

Do lado do código aberto, é principalmente a *Open Source Initiative* (OSI), fundada em 1998 por Eric S. Raymond e Bruce Perens, que trata das questões de licenciamento. Ela também desenvolveu um procedimento padronizado para verificar as licenças de software quanto à conformidade com sua *Definição de Código Aberto*. Atualmente, mais de 80 licenças reconhecidas de código aberto podem ser encontradas no site da OSI.

Há também licenças como a "OSI-approved", que contradizem explicitamente o princípio do copyleft, especialmente o grupo de *licenças BSD*. O *Berkeley Software Distribution* (BSD) é uma variante do sistema operacional Unix originalmente desenvolvido na Universidade de Berkeley, que mais tarde deu origem a projetos livres como *NetBSD*, *FreeBSD* e *OpenBSD*. As licenças subjacentes a esses projetos são freqüentemente consideradas *permissivas*. Em contraste com as licenças copyleft, elas não têm o objetivo de estabelecer os termos de uso das variantes modificadas. Em vez disso, a liberdade irrestrita tem o papel de ajudar o software a ser tão amplamente distribuído quanto possível, concedendo aos seus editores o poder único de decisão sobre o que fazer com as edições — se, por exemplo, elas também serão lançadas ou se serão tratadas como produtos de código fechado e distribuídas comercialmente.

A *Licença BSD de 2 Cláusulas*, também chamada *Licença BSD Simplificada* ou *Licença FreeBSD*, demonstra o quanto essas licenças permissivas podem ser sucintas. Além da cláusula padrão de responsabilidade, que protege os desenvolvedores contra pedidos de indenização decorrentes de danos causados pelo software, a licença consiste apenas nas duas regras a seguir:

> Redistribuição e uso em formas fonte e binária, com ou sem modificação, são permitidos desde que as seguintes condições sejam atendidas:
>
> 1. Redistribuições de código fonte devem manter o aviso de direitos autorais acima, esta lista de condições e o seguinte aviso legal.
> 2. Redistribuições em forma binária devem reproduzir o aviso de direitos autorais acima, esta lista de condições e o seguinte aviso legal na documentação e/ou outros materiais fornecidos com a distribuição.

Creative Commons

O bem-sucedido conceito de desenvolvimento do FLOSS e o progresso tecnológico associado levaram a tentativas de transferir o princípio do código aberto para outras áreas não técnicas. A preparação e a disponibilização de conhecimentos, bem como a cooperação criativa na resolução de tarefas complexas, são agora considerados como prova do princípio do código aberto, ampliado e relacionado ao conteúdo.

Isso levou à necessidade de criar bases sólidas igualmente nessas áreas, permitindo que os resultados do trabalho pudessem ser compartilhados e processados. Como as licenças de software disponíveis não eram adequadas a isso, houve inúmeras tentativas de converter em licenças igualmente úteis os requisitos específicos de áreas que vão desde a produção científica até obras de arte digitalizadas “no espírito do código aberto”.

Atualmente, a iniciativa mais importante desse tipo é de longe a *Creative Commons* (CC), que resume as suas preocupações da seguinte forma:

> A Creative Commons é uma organização global sem fins lucrativos que permite o compartilhamento e a reutilização da criatividade e do conhecimento através da disponibilização de ferramentas legais gratuitas.
>
> — https://creativecommons.org/faq/#what-is-creative-commons-and-what-do-you-do

Com a Creative Commons, o foco da atribuição de direitos deixa de ser o distribuidor e volta a ser o autor. Um exemplo: no processo editorial tradicional, um autor normalmente transfere todos os direitos de publicação (impressão, tradução etc.) para um editor, que por sua vez assegura a melhor distribuição possível da obra. Atualmente, os novos canais de distribuição possibilitados pela internet colocam o autor em posição de exercer ele próprio muitos desses direitos editoriais e de decidir por si mesmo como seu trabalho pode ser utilizado. A Creative Commons oferece a oportunidade de determinar isso de forma simples e legalmente confiável, mas ela quer mais: os autores são encorajados a tornar suas obras disponíveis como contribuição para um processo generalizado de troca e cooperação. Ao contrário do copyright tradicional, que dá ao autor todos os direitos que ele pode transferir para outras pessoas caso necessário, a Creative Commons adota a postura oposta: o autor disponibiliza sua obra à comunidade, mas pode escolher, dentre vários aspectos, aqueles que precisam ser levados em consideração quando a obra for utilizada — quanto mais aspectos forem escolhidos, mais restritiva será a licença.

E assim, o princípio “Escolha uma Licença” da CC pede que o autor defina passo a passo as propriedades individuais para gerar a licença recomendada, que o autor pode então incorporar à sua obra na forma de texto e ícone.

Para entender melhor, eis uma visão geral das seis possíveis combinações e licenças oferecidas

pela CC:

CC BY (“Atribuição”)

A licença livre que permite a qualquer pessoa editar e distribuir a obra, desde que cite o autor.

CC BY-SA (“Atribuição-CompartilhaIgual”)

Como a CC BY, exceto porque a obra modificada só pode ser distribuída sob a mesma licença. O princípio do copyleft permanece, porque a licença também é “herdada” aqui.

CC BY-ND (“Atribuição-SemDerivações”)

Como a CC BY, exceto porque a obra só pode ser repassada sem modificações.

CC BY-NC (“Atribuição-NãoComercial”)

A obra pode ser editada e distribuída com citação do autor, mas somente para fins não comerciais.

CC BY-NC-SA (“Atribuição-NãoComercial-CompartilhaIgual”)

Como a BY-NC, exceto porque a obra só pode ser compartilhada sob as mesmas condições (ou seja, uma licença semelhante à copyleft).

CC BY-NC-ND (“Atribuição-NãoComercial-SemDerivações”)

A licença mais restritiva de todas: a distribuição da obra é permitida com citação do autor, mas somente inalterada e para fins não comerciais.

Modelos de negócios em Open Source

Em retrospectiva, o trunfo da FLOSS é representar um movimento popular de tecnófilos idealistas que, livres de restrições econômicas e dependências monetárias, põem seu trabalho a serviço do público em geral. Ao mesmo tempo, empresas que valem bilhões foram criadas no ambiente FLOSS; poderíamos citar como exemplo a empresa americana *Red Hat*, fundada em 1993 com vendas anuais de mais de 3 bilhões de dólares (2018) e assumida pela gigante de TI IBM em 2018.

Então, vamos dar uma olhada na tensão entre a distribuição livre e quase sempre gratuita de software de alta qualidade e os modelos de negócios para seus criadores, porque uma coisa deve ficar clara: os inúmeros desenvolvedores altamente qualificados de software livre também precisam ganhar dinheiro, e o ambiente FLOSS original, puramente não-comercial, deve, portanto, desenvolver modelos de negócios sustentáveis para preservar seu próprio cosmos.

Uma abordagem comum, especialmente em projetos maiores em fase inicial, é o chamado *crowdfunding*, ou seja, a coleta de doações monetárias através de uma plataforma como o *Kickstarter*. Em troca, os doadores recebem um bônus predefinido dos desenvolvedores no caso

de sucesso, ou seja, se os objetivos previamente definidos forem alcançados, seja um acesso ilimitado ao produto ou recursos exclusivos.

Outra abordagem é a *licença dual*: o software livre é oferecido em paralelo sob uma licença mais restritiva ou mesmo proprietária, o que por sua vez garante ao cliente serviços mais extensos (tempos de resposta em caso de erros, atualizações, versões para certas plataformas etc.). Um exemplo dentre muitos é o *ownCloud*, que está sendo desenvolvido sob a GPL e oferece aos clientes empresariais uma “Business Edition” sob uma licença proprietária.

Tomemos também o ownCloud como exemplo de outro modelo de negócio amplamente difundido de FLOSS: serviços profissionais. Muitas empresas carecem do conhecimento técnico interno necessário para configurar e operar softwares complexos e críticos de forma confiável e, sobretudo, segura. Assim, elas contratam serviços profissionais, como consultoria, manutenção ou assistência técnica, diretamente do fabricante. As questões de responsabilidade também têm seu papel nessa decisão, já que a empresa transfere os riscos da operação para o fabricante.

Se um software consegue se tornar bem-sucedido e popular em seu campo, existem possibilidades periféricas de monetização, como o merchandising ou certificados adquiridos pelos clientes para indicar seu status especial no uso daquele software. A plataforma de aprendizagem *Moodle* oferece a certificação de instrutores, oficializando seus conhecimentos para clientes potenciais, por exemplo, e este é apenas um exemplo dentre inúmeros outros.

O *Software como Serviço* (Software as a Service ou SaaS) é outro modelo de negócios, especialmente para tecnologias baseadas na web. Aqui, um provedor de nuvem executa um software como uma Gestão de Relacionamento com o Cliente (Customer Relationship Management ou CRM) ou um Sistema de Gerenciamento de Conteúdo (Content Management System ou CMS) em seus servidores e oferece aos seus clientes acesso ao aplicativo instalado. Isso evita que o cliente precise se preocupar com a instalação e a manutenção do software. Em contrapartida, o cliente paga pelo uso do software de acordo com diversos parâmetros, como por exemplo o número de usuários. A disponibilidade e a segurança têm um papel importante, sendo fatores críticos no mundo empresarial.

Por último, mas não menos importante, o modelo de desenvolvimento por encomenda de extensões específicas em software livre para um cliente é particularmente comum em projetos menores. Normalmente, cabe ao cliente decidir o que fazer com essas extensões, ou seja, se ele também as libera ou se as mantêm privadas como parte de seu próprio modelo de negócios.

Uma coisa deve ficar clara: embora o software livre esteja geralmente disponível gratuitamente, muitos modelos de negócios foram criados nesse ambiente, e eles são constantemente modificados e ampliados por inúmeros freelancers e empresas em todo o mundo de maneira muito criativa, o que, em última análise, também garante a continuidade da existência de todo o

movimento FLOSS.

Exercícios Guiados

1. Quais são — de forma resumida — as “quatro liberdades” definidas por Richard Stallman e a Free Software Foundation?

liberdade 0	
liberdade 1	
liberdade 2	
liberdade 3	

2. O que significa a abreviatura FLOSS?

3. Você desenvolveu um software livre e quer garantir que o software em si, mas também todas as futuras criações que se baseiem nele, também sejam livres. Qual licença deve escolher?

CC BY	
GPL versão 3	
Licença BSD de 2 Cláusulas	
LGPL	

4. Quais das seguintes licenças você considera permissivas e quais considera copyleft?

Licença BSD Simplificada	
GPL versão 3	
CC BY	
CC BY-SA	

5. Você escreveu um aplicativo web e o publicou sob uma licença livre. Como você pode ganhar dinheiro com seu produto? Cite três possibilidades.

Exercícios Exploratórios

1. Sob qual licença (incluindo a versão) estão disponíveis os seguintes aplicativos?

Apache HTTP Server	
MySQL Community Server	
Artigos da Wikipedia	
Mozilla Firefox	
GIMP	

2. Você deseja publicar seu software sob a GNU GPL v3. Qual o procedimento a seguir?

3. Você escreveu software proprietário e gostaria de combiná-lo com software livre sob a GPL versão 3. Existe essa possibilidade? O que deve ser levado em conta?

4. Por que a Free Software Foundation lançou a *GNU Affero General Public License* (GNU AGPL) como suplemento à GNU GPL?

5. Dê três exemplos de software livre que também é oferecido em “Edição Empresarial”, ou seja, em versão paga.

Sumário

Nesta lição você aprendeu:

- Semelhanças e diferenças entre software livre e de código aberto (FLOSS)
- Licenças FLOSS, sua importância e problemas
- Copyleft versus licenças permissivas
- Modelos de negócio FLOSS

Respostas aos Exercícios Guiados

1. Quais são — de forma resumida — as “quatro liberdades” definidas por Richard Stallman e a Free Software Foundation?

liberdade 0	executar o software
liberdade 1	estudar e modificar o software (código fonte)
liberdade 2	distribuir o software
liberdade 3	distribuir o software modificado

2. O que significa a abreviatura FLOSS?

   Free/Libre Open Source Software

3. Você desenvolveu um software livre e quer garantir que o software em si, mas também todas as futuras criações que se baseiem nele, também sejam livres. Qual licença deve escolher?

CC BY	
GPL versão 3	X
Licença BSD de 2 Cláusulas	
LGPL	

4. Quais das seguintes licenças você considera permissivas e quais considera copyleft?

Licença BSD Simplificada	permissiva
GPL versão 3	copyleft
CC BY	permissiva
CC BY-SA	copyleft

5. Você escreveu um aplicativo web e o publicou sob uma licença livre. Como você pode ganhar dinheiro com seu produto? Cite três possibilidades.
   - Licença dual, ou seja, oferecendo uma “Edição Empresarial” paga
   - Oferecendo hospedagem, serviços e suporte
   - Desenvolvendo extensões proprietárias para os clientes

Respostas aos Exercícios Exploratórios

1. Sob qual licença (incluindo a versão) estão disponíveis os seguintes aplicativos?

Apache HTTP Server	Apache License 2.0
MySQL Community Server	GPL 2.0
Wikipedia articles (English)	Licença Creative Commons Atribuição-CompartilhaIgual (CC-BY-SA)
Mozilla Firefox	Mozilla Public License 2.0
GIMP	LGPL 3

2. Você deseja publicar seu software sob a GNU GPL v3. Qual o procedimento a seguir?
   - Se necessário, proteger-se contra o empregador com uma renúncia aos direitos autorais, por exemplo, para poder especificar a licença.
   - Adicionar um aviso de direitos autorais a cada arquivo.
   - Adicionar um arquivo chamado `COPYING` com o texto completo da licença de seu software.
   - Adicionar uma referência à licença a cada arquivo.
3. Você escreveu software proprietário e gostaria de combiná-lo com software livre sob a GPL versão 3. Existe essa possibilidade? O que deve ser levado em conta?

   As Questões Frequentes da Free Software Foundation trazem essa informação: Desde que seu software proprietário e o software livre permaneçam separados, a combinação é possível. Porém, você deve garantir que essa separação seja tecnicamente assegurada e reconhecível pelos usuários. Se você integrar o software livre de tal maneira que ele se torne parte integrante de seu produto, deve publicar também o produto sob a GPL de acordo com o princípio do copyleft.

4. Por que a Free Software Foundation lançou a *GNU Affero General Public License* (GNU AGPL) como suplemento à GNU GPL?

   A GNU AGPL vem preencher uma lacuna de licenciamento que ocorre especialmente com software livre hospedado em um servidor: se um desenvolvedor fizer alterações no software, ele não é obrigado, sob a GPL, a tornar essas alterações acessíveis, já que ele permite acesso ao programa, mas não "redistribui" o programa no sentido definido pela GPL. A GNU AGPL, por outro lado, estipula que o software deve ser disponibilizado para download com todas as alterações.

5. Dê três exemplos de software livre que também é oferecido em “Edição Empresarial”, ou seja, em versão paga.

   MySQL, Zammad, Nextcloud

1.4 ICT Habilidades ICT e trabalhando no Linux

Referência ao LPI objectivo

Linux Essentials version 1.6, Exam 010, Objective 1.4

Peso

2

Áreas chave de conhecimento

- Habilidades no Desktop
- Chegando à linha de comando
- Usos industriais para Linux, computação em nuvem e virtualização

Segue uma lista parcial de arquivos, termos e utilitários utilizados

- Usando um navegador, questões de privacidade, opções de configuração, procurando na web e salvando conteúdo
- Terminal e Console
- Senhas
- Questões e ferramentas relacionadas à privacidade
- Uso de aplicações Open Source populares em apresentações e projetos

1.4 Lição 1

Certificação:	Linux Essentials
Versão:	1.6
Tópico:	1 A Comunidade Linux e uma carreira em open source
Objetivo:	1.4 Conhecimentos de TIC e o trabalho com o Linux
Lição:	1 de 1

Introdução

Houve um tempo em que trabalhar com Linux no desktop era considerado difícil, já que o sistema carecia de muitos dos aplicativos e ferramentas de configuração mais elaborados oferecidos por outros sistemas operacionais. Dito isso, era mais simples começar pelo desenvolvimento dos aplicativos mais essenciais em linha de comando e deixar as ferramentas gráficas mais complexas para mais tarde. No início, como o Linux foi inicialmente pensado para usuários mais avançados, isso não era um grande problema. Mas esses dias terminaram. Hoje em dia, os ambientes de desktop Linux estão bem mais maduros, não deixando nada a desejar no que diz respeito aos recursos e à facilidade de uso. Ainda assim, a linha de comando ainda é considerada uma ferramenta poderosa, empregada diariamente pelos usuários avançados. Nesta lição, vamos apresentar alguns conhecimentos básicos de desktop necessários para você saber escolher a melhor ferramenta para cada tarefa, começando com o acesso à linha de comando.

Interfaces de usuário Linux

Ao usar um sistema Linux, você interage com uma linha de comando ou com interfaces gráficas de usuário. Ambas as opções oferecem acesso a diversos aplicativos capazes de executar praticamente qualquer tarefa no computador. Embora o objetivo 1.2 já tenha apresentado uma série de aplicativos comumente usados, vamos começar esta lição com um olhar mais atento para os ambientes desktop, as formas de acessar o terminal e as ferramentas usadas para apresentações e gerenciamento de projetos.

Ambientes desktop

A abordagem do Linux é modular, ou seja, diferentes partes do sistema são desenvolvidas por diferentes projetos e desenvolvedores, que dão conta de uma necessidade ou de um objetivo específico. Por essa razão, existem várias opções de ambientes desktop para escolher, bem como diversos gerenciadores de pacotes, e por isso o ambiente desktop padrão pode variar bastante entre as muitas distribuições existentes. Ao contrário dos sistemas operacionais proprietários, como Windows e MacOS, nos quais os usuários ficam restritos ao ambiente desktop que vem com o SO, existe a possibilidade de se instalar diversos ambientes e escolher aquele que mais se adapta a você e a suas necessidades.

Basicamente, existem dois ambientes de trabalho principais no mundo Linux: *Gnome* e *KDE*. Ambos são bem completos, contam com uma grande comunidade e visam o mesmo propósito, mas com abordagens ligeiramente divergentes. Para resumir, o Gnome busca seguir o princípio KISS (“keep it simple stupid”, ou “mantenha simples, estúpido”, em português), com aplicações simplificadas e limpas. Por sua vez, o KDE tem outra perspectiva, oferecendo um leque maior de aplicativos e oferecendo ao usuário a possibilidade de alterar cada uma das configurações do ambiente.

Enquanto os aplicativos Gnome baseiam-se no kit de ferramentas GTK (escrito na linguagem C), os aplicativos KDE utilizam a biblioteca Qt (escrita em C++). Um dos aspectos mais práticos de se escrever programas usando o mesmo kit de ferramentas gráficas é que eles ficam com uma aparência e usabilidade semelhantes, o que proporciona um sentido de coesão ao usuário durante sua experiência. Outra característica importante é que, quando empregamos a mesma biblioteca gráfica compartilhada em diversos aplicativos de uso frequente, economizamos espaço na memória, além de reduzirmos o tempo de carregamento depois de a biblioteca ter sido carregada pela primeira vez.

Como abrir a linha de comando

Para nós, um dos aplicativos mais importantes é o emulador de terminal gráfico. São chamados de emuladores de terminal porque realmente emulam, em um ambiente gráfico, os antigos terminais

seriais (muitas vezes máquinas de teletipo) que eram na verdade clientes conectados a uma máquina remota na qual a computação acontecia de fato. Essas máquinas eram, na verdade, computadores muito simples e sem gráficos, usados nas primeiras versões do Unix.

No Gnome, esse aplicativo se chama *Gnome Terminal*, enquanto no KDE ele é conhecido como *Konsole*. Mas há muitas outras opções disponíveis, tais como o *Xterm*. Estes aplicativos são uma forma de termos acesso a um ambiente de linha de comando e, assim, podermos interagir com um shell.

Assim, procure o aplicativo de terminal no menu de programas da distribuição de sua escolha. Tirando pequenas diferenças entre eles, todos esses aplicativos oferecem o necessário para você ganhar confiança no uso da linha de comando.

Outra maneira de entrar no terminal é usar o TTY virtual. Para encontrá-lo, pressione `Ctrl` + `Alt` + `F#`. Leia `F#` como uma das teclas de função de 1 a 7, por exemplo. É provável que algumas das combinações iniciais estejam associadas ao gerenciador de sessão ou ao ambiente gráfico. As outras irão exibir um prompt pedindo seu nome de login, como no exemplo abaixo:

```
Ubuntu 18.10 arrelia tty3
arrelia login:
```

O `arrelia`, neste caso, é o hostname da máquina, e `tty3` é o terminal disponível a partir da combinação de teclas descrita acima, mais a tecla `F3`, como em `Ctrl` + `Alt` + `F3`.

Depois de inserir seu login e senha, você finalmente entrará em um terminal, mas sem nenhum ambiente gráfico, de modo que não será possível usar o mouse ou executar aplicações gráficas sem antes iniciar uma sessão X, ou Wayland. Mas isso está além do escopo desta lição.

Apresentações e projetos

A ferramenta mais importante para apresentações no Linux é o *LibreOffice Impress*. Ele faz parte do pacote de escritório *LibreOffice*. Pense no LibreOffice como um substituto de código aberto para o *Microsoft Office*. Ele pode inclusive abrir e salvar arquivos PPT e PPTX, nativos do *Powerpoint*. Mas apesar disso, é recomendável dar preferência ao formato nativo ODP Impress. O ODP é parte do *Open Document Format*, um padrão internacional para esse tipo de arquivo. Isso é especialmente importante se você deseja manter seu documento acessível por muitos anos sem se preocupar com problemas de compatibilidade. Por se tratar de um padrão aberto, qualquer pessoa pode implementar esse formato sem pagar royalties ou licenças. Isso também permite que você possa experimentar outros softwares de apresentação sem perder o acesso aos seus arquivos, já que há grande probabilidade de que eles sejam compatíveis com programas mais recentes.

Mas se você prefere código em vez de interfaces gráficas, tem um punhado de ferramentas à escolha. O *Beamer* é uma classe *LaTeX* capaz de criar apresentações de slides a partir de código LaTeX. O LaTeX em si é um sistema de composição de texto amplamente utilizado para escrever documentos científicos acadêmicos, sobretudo por sua capacidade de lidar com símbolos matemáticos complexos, que causam dificuldades para outros softwares. Se você está na universidade e precisa lidar com equações e outros problemas relacionados à matemática, o Beamer pode lhe poupar um tempo precioso.

A outra opção é o *Reveal.js*, um incrível pacote NPM (NPM é o gerenciador de pacotes padrão do NodeJS) que permite criar belas apresentações usando a web. Assim, se você entende de HTML e CSS, o Reveal.js contribuirá com a maior parte do JavaScript necessário para criar apresentações bonitas e interativas que se adaptam bem a qualquer resolução e tamanho de tela.

Por fim, se você procura um substituto para o *Microsoft Project*, sugerimos o *GanttProject* ou o *ProjectLibre*. Ambos são muito semelhantes ao seu equivalente proprietário e compatíveis com arquivos do Project.

Usos do Linux na indústria

O Linux é muito utilizado nas indústrias de software e internet. Sites como W3Techs relatam que cerca de 68% dos servidores de websites na Internet são alimentados por Unix, a maioria deles com Linux.

Essa grande proporção se deve não somente à natureza livre do Linux (tanto no sentido de grátis quanto no de liberdade de expressão), mas também por sua estabilidade, flexibilidade e desempenho. Essas características permitem que os fornecedores ofereçam seus serviços com menor custo e maior escalabilidade. Uma porção significativa dos sistemas Linux atualmente é executada na nuvem, seja no modelo IaaS (Infraestrutura como Serviço), PaaS (Plataforma como Serviço) ou SaaS (Software como Serviço).

O IaaS é uma maneira de compartilhar os recursos de um grande servidor oferecendo acesso a máquinas virtuais que são, na verdade, múltiplos sistemas operacionais rodando como convidados em uma máquina hospedeira sobre um importante software chamado *hipervisor*. O hipervisor é responsável por permitir que os SOs convidados possam rodar, segregando e gerenciando os recursos disponíveis na máquina hospedeira para esses convidados. Isso é o que nós chamamos *virtualização*. No modelo IaaS, você paga apenas pela fração de recursos usados por sua infraestrutura.

O Linux tem três hipervisores de código aberto bastante conhecidos: *Xen*, *KVM* e *VirtualBox*. O Xen é provavelmente o mais antigo deles. O KVM superou o Xen como o hipervisor Linux mais relevante. Seu desenvolvimento é patrocinado pela RedHat e ele é usado também por outros

atores da indústria, tanto em serviços de nuvem pública quanto em configurações de nuvem privada. O VirtualBox pertence à Oracle desde que esta adquiriu a Sun Microsystems e é geralmente escolhido pelos usuários finais devido à sua facilidade de uso e de administração.

Já o PaaS e o SaaS incrementam o modelo IaaS, tanto tecnicamente quanto conceitualmente. No PaaS, em vez de uma máquina virtual, os usuários têm acesso a uma plataforma na qual é possível implantar e executar seu aplicativo. A ideia é aliviar a carga de se lidar com tarefas de administração de sistema e atualizações de sistema operacional. O Heroku é um exemplo comum de PaaS no qual o código do programa pode ser simplesmente executado sem que seja preciso cuidar dos containers subjacentes e máquinas virtuais.

E finalmente, o SaaS é o modelo em que geralmente se paga por uma assinatura para simplesmente usar um software sem se preocupar com mais nada. O_Dropbox_ e o *Salesforce* são dois bons exemplos de SaaS. A maioria desses serviços é acessada através de um navegador web.

Um projeto como o *OpenStack* é uma coleção de software de código aberto que pode fazer uso de diferentes hipervisores e outras ferramentas a fim de oferecer um ambiente de nuvem IaaS completo no local, aproveitando o poder do cluster de computadores em seu próprio centro de processamento de dados. No entanto, a configuração dessa infraestrutura não é uma tarefa trivial.

Problemas de privacidade ao se usar a internet

O navegador web é um programa fundamental em qualquer desktop de nossa época, mas algumas pessoas ainda não sabem como utilizá-los de maneira segura. Embora mais e mais serviços sejam acessados através de um navegador, quase todas as ações feitas no navegador são rastreadas e analisadas por diferentes intermediários. Proteger o acesso aos serviços de Internet e evitar o rastreamento são aspectos importantes para um uso seguro da Internet.

Rastreamento de cookies

Vamos supor que você visitou uma loja virtual, selecionou um produto que queria e o colocou em seu carrinho. Mas, no último segundo, decidiu tirar um tempo para refletir e decidir se realmente precisava daquilo. Depois de um tempinho, você começa a ver anúncios daquele mesmo produto em todos os cantos da web. Ao clicar nesses anúncios, você é imediatamente encaminhado à página do produto naquela mesma loja. Não é incomum que os produtos colocados no carrinho ainda estejam lá, esperando que você se decida a adquiri-los. Já se perguntou como eles fazem isso? Como mostram o anúncio certo em outra página web? A resposta a essas questões está no *rastreamento de cookies*.

Os Cookies são pequenos arquivos que um site pode salvar em seu computador para armazenar e recuperar algum tipo de informação que possa ser útil em sua navegação. Eles existem há muitos

anos e são uma das maneiras mais antigas de armazenar dados no lado do cliente. Um bom exemplo de seu uso são as identidades únicas para os carrinhos de compras. Dessa forma, se você voltar ao mesmo site em poucos dias, a loja pode lembrar-lhe os produtos que você colocou em seu carrinho durante a última visita e poupar-lhe o tempo de procurá-los novamente.

Até aí tudo bem, já que o site está oferecendo um recurso útil e não compartilhando seus dados com terceiros. Mas e quanto aos anúncios exibidos enquanto você navega em outras páginas da web? É aí que entram as redes de anúncios. As redes de anúncios são empresas que oferecem anúncios para lojas virtuais como a do nosso exemplo, de um lado, e monetização para websites, do outro. Os criadores de conteúdo, como blogueiros, por exemplo, podem disponibilizar espaço para essas redes de anúncios em seu blog em troca de uma comissão relacionada às vendas geradas pelo anúncio.

Mas como eles sabem qual produto exibir no anúncio? Isso normalmente é feito com um cookie da rede de anúncios que também é salvo no momento em que você visitou ou buscou um produto determinado na loja virtual. Ao fazer isso, a rede consegue recuperar as informações desse cookie nos sites em que a rede anuncia, fazendo a correlação com os produtos em que você se mostrou interessado. Essa é uma das maneiras mais comuns de se rastrear alguém pela Internet. Usamos o exemplo de uma loja online para dar uma explicação mais tangível, mas as plataformas de mídia social fazem o mesmo com seus botões de “Curtir” ou “Compartilhar”.

Uma maneira de se livrar desse efeito é não permitir que sites de terceiros armazenem cookies em seu navegador. Dessa forma, somente o site que você visita pode armazenar seus cookies. Mas esteja ciente de que alguns recursos “legítimos” podem não funcionar bem se você fizer isso, já que muitos sites atualmente dependem de serviços de terceiros para funcionar. Então, você pode procurar por um gerenciador de cookies no repositório de complementos do seu navegador, a fim de ter um controle fino sobre os cookies que serão armazenados em sua máquina.

Não Rastrear (DNT)

Outro equívoco comum está relacionado a uma determinada configuração de browser conhecida como DNT. Esse é um acrônimo de “Do Not Track” (Não Rastrear) e pode ser ativado basicamente em qualquer navegador atual. Como no caso do modo privado, não é difícil encontrar pessoas que acreditam que não serão rastreadas se ativarem essa configuração. Infelizmente, isso nem sempre é verdade. Atualmente, o DNT é apenas uma forma de dizer aos websites visitados que você não quer que eles o rastreiem, mas na verdade são eles que vão decidir se vão respeitar sua escolha ou não. Em outras palavras, o DNT é uma maneira de pedir para ser excluído do rastreamento dos websites, mas não há garantia de que esse pedido será atendido.

Tecnicamente, isso é feito simplesmente emitindo uma flag extra no cabeçalho do protocolo da requisição HTTP (`DNT: 1`) ao se requisitar dados de um servidor web. Se quiser saber mais sobre

este tópico, o site https://allaboutdnt.com é um bom ponto de partida.

Janelas “anônimas”

Você deve ter notado as aspas no título acima. Isso porque essas janelas não são tão anônimas quanto a maioria das pessoas pensa. Os nomes podem variar: elas podem ser chamadas de “modo privado”, “incógnito” ou “anônimo”, dependendo do navegador que você estiver usando.

No Firefox, é fácil ativar esse modo pressionando as teclas `Ctrl` + `Shift` + `P`. No Chrome, basta pressionar `Ctrl` + `Shift` + `N`. O que ele realmente faz é abrir uma nova sessão, a qual normalmente não compartilha nenhuma configuração ou dados do seu perfil padrão. Quando você fecha a janela privada, o navegador exclui automaticamente todos os dados gerados por essa sessão, não deixando vestígios no computador usado. Assim, nenhum dado pessoal, como histórico, senhas ou cookies, serão armazenados nesse computador.

Porém, muita gente interpreta erroneamente esse conceito, acreditando que é possível navegar na internet de forma anônima, o que não é inteiramente verdade. Uma das coisas que o modo privado ou incógnito faz é evitar aquilo que chamamos de rastreamento de cookies. Quando você visita um site, ele pode armazenar um pequeno arquivo em seu computador contendo um identificador que pode ser usado para rastreá-lo. A menos que você configure seu navegador para não aceitar cookies de terceiros, as redes de anúncios ou outras empresas podem armazenar e recuperar esse identificador e rastrear sua navegação na web. Mas como os cookies armazenados em uma sessão de modo privado são excluídos logo após o encerramento da sessão, essas informações são perdidas para sempre.

Além disso, os sites e outros agentes da internet ainda podem usar muitas outras técnicas para rastrear suas atividades. Portanto, o modo privado permite um certo nível de anonimato, mas é completamente privado apenas no computador que está sendo utilizado. Se você precisa acessar sua conta de email ou site bancário a partir de um computador público, por exemplo em um aeroporto ou hotel, não deve deixar de utilizar o modo privado do navegador. Em outras situações, pode haver benefícios, mas é preciso saber exatamente quais riscos estão sendo evitados e quais permanecem. Sempre que utilizar um computador acessível ao público, esteja ciente de que outras ameaças à segurança, como malware ou keyloggers, podem estar presentes. Seja prudente sempre que inserir informações pessoais, como nomes de usuário e senhas, nesses computadores, ou quando baixar ou copiar dados confidenciais.

Escolha a senha certa

Uma das situações mais difíceis enfrentadas por um usuário é ter de escolher uma senha segura para os serviços que utiliza. Você certamente já ouviu dizer que não se deve usar combinações comuns como `qwerty`, `123456` ou `654321`, nem números fáceis de adivinhar como seu

aniversário (ou de um parente) ou código postal. A razão para isso é que essas combinações são muito óbvias, certamente as primeiras que um invasor vai experimentar a fim de ganhar acesso à sua conta.

Existem técnicas conhecidas para criar uma senha segura. Uma das mais famosas é inventar uma frase que lembre aquele serviço e pegar as primeiras letras de cada palavra. Vamos supor que eu quero criar uma boa senha para o Facebook, por exemplo. Nesse caso, eu poderia criar uma frase como "Eu ficaria feliz se tivesse 1000 amigos como o Claudemir". Pegamos a primeira letra de cada palavra e a senha final seria "Effst1000acoC". Isso resultaria em uma senha de 13 caracteres, longa o suficiente para ser difícil de adivinhar e, ao mesmo tempo, fácil de decorar (desde que eu seja capaz de me lembrar da frase e do “algoritmo” para recuperar a senha).

Frases são mais fáceis de lembrar do que senhas, mas até esse método tem suas limitações. Temos de criar senhas para uma porção de serviços hoje em dia, e como as usamos com frequências diferentes, acaba ficando muito difícil lembrar de todas as frases no momento em que precisamos delas. Então, o que podemos fazer? Você pode responder que a coisa mais sábia a fazer neste caso é criar um punhado de boas senhas e reutilizá-las em serviços similares, certo?

Infelizmente, essa também não é uma boa ideia. Você provavelmente também já ouviu dizer que não deve reutilizar a mesma senha para diferentes serviços. O problema de se fazer isso é que um serviço específico pode vazar sua senha (sim, acontece bastante) e qualquer pessoa que tenha acesso a ela tentará usar a mesma combinação de email e senha em outros serviços populares da Internet, na esperança de que você tenha feito exatamente isso: reciclado senhas. E adivinhe? Caso esse palpite esteja certo, você acabará tendo um problema não apenas em um serviço, mas em vários. E pode acreditar: sempre achamos que não vai acontecer conosco, até ser tarde demais.

Então, o que podemos fazer para nos proteger? Uma das abordagens mais seguras disponíveis hoje é usar um *gerenciador de senhas*. Os gerenciadores de senhas são programas que essencialmente armazenam todas as suas senhas e nomes de usuário em um formato criptografado que pode ser descriptografado por uma senha mestra. Dessa forma, você só precisa se lembrar de uma boa senha, já que o gerenciador cuidará de manter todas as outras em segurança.

O *KeePass* é um dos gerenciadores de senha open source mais famosos e ricos em recursos do mercado. Ele armazena suas senhas em um arquivo criptografado dentro de seu sistema de arquivos. O fato de ser open source é uma questão importante para este tipo de software, pois isso garante que não haverá um uso espúrio de seus dados, já que qualquer desenvolvedor pode auditar o código e saber exatamente como funciona. Isso proporciona um nível de transparência impossível de alcançar com código proprietário. O KeePass tem versões para a maioria dos sistemas operacionais, incluindo Windows, Linux e macOS, bem como para os sistemas móveis, como iOS e Android. Também inclui um sistema de plugins capaz de estender sua funcionalidade

para bem além do padrão.

O *Bitwarden* é outra solução de código aberto com uma abordagem semelhante, mas em vez de armazenar seus dados em um arquivo, ele lança mão de um servidor na nuvem. Dessa forma, é mais fácil manter todos os seus dispositivos sincronizados e suas senhas facilmente acessíveis através da web. O *Bitwarden* é um dos poucos projetos em que não só os clientes, mas também o servidor na nuvem estão disponíveis como software de código aberto. Isso significa que é possível hospedar sua própria versão do Bitwarden e torná-la disponível para qualquer pessoa, como a sua família ou os funcionários da sua empresa. Isso lhe dará flexibilidade, mas também total controle sobre como essas senhas são armazenadas e utilizadas.

Uma das coisas mais importantes a se ter em mente ao usar um gerenciador de senhas é criar uma senha aleatória para cada serviço diferente, uma vez que você não precisará decorá-las. Seria inútil usar um gerenciador de senhas para armazenar senhas recicladas ou facilmente adivinháveis. Assim, a maioria deles inclui um gerador de senhas aleatórias que cria senhas fortíssimas para você.

Criptografia

Sempre que fazemos transferências ou armazenamos dados, é preciso tomar precauções para garantir que terceiros não possam ter acesso a eles. Os dados transferidos pela internet passam por uma série de roteadores e redes que podem ser acessados por terceiros. Do mesmo modo, os dados armazenados em meios físicos podem ser lidos por qualquer pessoa que entre em posse deles. Para evitar esse tipo de acesso, as informações confidenciais devem ser criptografadas antes de sair de um dispositivo informático.

TLS

O *Transport Layer Security* (TLS) é um protocolo que aumenta a segurança das conexões de rede por meio da criptografia. O TLS é o sucessor do *Secure Sockets Layer* (SSL), que se tornou obsoleto devido a falhas graves. O TLS também evoluiu algumas vezes para se adaptar e tornar-se mais seguro, por isso a versão mais atual é a 1.3. Ele garante tanto a privacidade quanto a autenticidade, graças ao uso de criptografia simétrica e de chave pública. Isso significa que, uma vez em uso, é possível ter certeza de que ninguém será capaz de interceptar ou alterar sua comunicação com aquele servidor durante aquela sessão.

A lição mais importante aqui é saber reconhecer se um website é confiável. Procure sempre pelo símbolo de “cadeado” na barra de endereços do navegador. Se desejar, você pode clicar nele para inspecionar o certificado, que desempenha um papel importante no protocolo HTTPS.

O TLS é usado no protocolo HTTPS (*HTTP sobre TLS*) para possibilitar o envio de dados

confidenciais (como o número do seu cartão de crédito) através da web. O funcionamento do TLS vai muito além do objetivo deste artigo, mas você pode encontrar mais informações em Wikipedia e em Mozilla wiki.

Criptografia de arquivos e emails com GnuPG

Existem muitas ferramentas para proteger emails, mas uma das mais importantes é certamente o *GnuPG*. GnuPG significa *GNU Privacy Guard*; trata-se de uma implementação open source do *OpenPGP*, um padrão internacional codificado no RFC 4880.

O GnuPG pode ser usado para assinar, criptografar e descriptografar textos, emails, arquivos, diretórios e até mesmo partições de disco inteiras. Ele utiliza criptografia de chave pública e está amplamente disponível. Em poucas palavras, o GnuPG cria um par de arquivos que contêm suas chaves públicas e privadas. Como o nome indica, a chave pública pode ser disponibilizada para qualquer pessoa e a chave privada precisa ser mantida em segredo. As pessoas usam sua chave pública para criptografar dados que somente sua chave privada será capaz de decodificar.

A chave privada também pode ser usada para assinar qualquer arquivo ou email, que poderão ser validados com a chave pública correspondente. Essa assinatura digital funciona de forma análoga a uma assinatura do mundo real. A partir do momento em que você é o único a conhecer sua chave privada, o receptor pode ter a certeza de você é o autor. Ao lançar mão da funcionalidade de hash criptográfico, o GnuPG também garante que não foram feitas alterações após a assinatura, já que qualquer mudança no conteúdo a invalidaria.

O GnuPG é uma ferramenta muito poderosa e, em certa medida, também complexa. Para saber mais, visite website e Archlinux wiki (a wiki Archlinux é uma boa fonte de informação, mesmo que você não use Archlinux).

Criptografia de disco

Uma boa maneira de garantir a segurança de seus dados é criptografar inteiramente o disco ou partição. Existem muitos softwares open source que podem ser usados com esse fim. Seu funcionamento e o nível de criptografia que oferecem também varia bastante. Existem dois métodos básicos: criptografia de dispositivos *empilhada* e *de bloco*.

As soluções de criptografia empilhada no sistema de arquivos são implementadas por cima do sistema de arquivos existente. Ao usar esse método, os arquivos e diretórios são criptografados antes de serem armazenados no sistema de arquivos e descriptografados depois de lidos. Isso significa que os arquivos são armazenados no sistema de arquivos hospedeiro em um formato criptografado (ou seja, seu conteúdo, e geralmente também os nomes de arquivos/pastas, são substituídos por dados aparentemente aleatórios), mas tirando isso, eles ainda existem no sistema

de arquivos sem criptografia, como arquivos normais, links simbólicos, hardlinks etc.

Por outro lado, a criptografia de dispositivos de bloco ocorre abaixo da camada do sistema de arquivos, garantindo que tudo o que é escrito num dispositivo de bloco é criptografado. Se olharmos para o bloco offline, ele vai parecer uma grande massa de dados aleatórios, e não será nem mesmo possível dizer qual é o tipo de sistema de arquivos sem antes descriptografá-lo. Ou seja, não dá para saber o que é um arquivo ou diretório, nem seu tamanho e de que tipo de dados se trata, porque os metadados, a estrutura de diretórios e as permissões também são criptografados.

Ambos os métodos têm seus pontos positivos e negativos. Dentre todas as opções disponíveis, sugerimos dar uma olhada no *dm-crypt*, que é o padrão de fato para a criptografia de bloco em sistemas Linux, sendo nativo no kernel. Pode ser usado com a extensão *LUKS* (*Linux Unified Key Setup*), uma especificação que implementa um padrão independente de plataforma, para uso com diversas ferramentas.

Se quiser experimentar um método de criptografia empilhada, sugerimos o *EncFS*, que é provavelmente a maneira mais simples de proteger dados no Linux, já que não requer privilégios de root para ser implementado e pode rodar em um sistema de arquivos existente sem modificações.

Finalmente, se você precisa acessar dados em várias plataformas, confira o Veracrypt. Ele é o sucessor do Truecrypt e permite a criação de mídia e arquivos criptografados, que podem ser usados no Linux, bem como em macOS e Windows.

Exercícios Guiados

1. Você deve usar uma “janela anônima” em seu navegador quando quer:

Navegar na internet de maneira completamente anônima	
Não deixar traços no computador que está usando	
Ativar o TLS para evitar o rastreamento de cookies	
Para usar o DNT	
Para usar criptografia durante a transmissão de dados	

2. O que é OpenStack?

Um projeto que permite a criação de um IaaS privado	
Um projeto que permite a criação de um PaaS privado	
Um projeto que permite a criação de um SaaS privado	
Um hipervisor	
Um gerenciador de senhas de código aberto	

3. Quais das opções abaixo são softwares válidos para criptografia de discos?

RevealJS, EncFS e dm-crypt	
dm-crypt e KeePass	
EncFS e Bitwarden	
EncFS e dm-crypt	
TLS e dm-crypt	

4. Selecione verdadeiro ou falso para a criptografia de dispositivos com dm-crypt:

Os arquivos são criptografados antes de serem escritos no disco	
O sistema de arquivos inteiro se torna uma massa criptografada	
Apenas os arquivos e diretórios são criptografados, e não os links simbólicos	
Não requer acesso de root	
É uma criptografia de dispositivo de bloco	

5. O Beamer é:

Um mecanismo de criptografia	
Um hipervisor	
Um programa de virtualização	
Um componente do OpenStack	
Uma ferramenta de apresentações em LaTeX	

Exercícios Exploratórios

1. A maioria das distribuições vem com o Firefox instalado por padrão (se não for o seu caso, será preciso instalá-lo antes). Vamos instalar uma extensão para o Firefox chamada *Lightbeam*. Para isso, você pode pressionar `Ctrl` + `Shift` + `A` e buscar por "Lightbeam" no campo de busca que aparece na aba aberta, ou visitar a página da extensão com o Firefox e clicar no botão "Instalar": https://addons.mozilla.org/en-GB/firefox/addon/lightbeam-3-0/. Em seguida, inicie a extensão clicando no ícone e comece a visitar páginas web em outras abas para ver o que acontece.
2. Qual a coisa mais importante ao se usar um gerenciador de senhas?

3. Use seu navegador para visitar https://haveibeenpwned.com/. Descubra qual a finalidade do site e confira se seu endereço de email foi incluído em algum vazamento de dados.

Sumário

O terminal é uma maneira poderosa de interagir com o sistema e existem muitas ferramentas úteis e maduras para se usar nesse tipo de ambiente. Para abrir o terminal, procure no menu de seu ambiente de desktop ou pressione `Ctrl` + `Alt` + `F#`.

O Linux é amplamente usado na indústria da tecnologia para oferecer serviços de IaaS, PaaS e SaaS. Existem três hipervisores principais que desempenham um papel importante no suporte desses serviços: Xen, KVM e Virtualbox.

O navegador é um programa essencial para a computação atualmente, mas é necessário entender como utilizá-lo com segurança. O DNT é somente uma maneira de informar ao site que você não deseja ser rastreado, mas não há garantia de que ele obedeça. As janelas anônimas são privadas somente em relação ao computador que você está usando, mas graças a isso elas podem ajudar a evitar o rastreamento de cookies.

O TLS pode criptografar suas comunicações na internet, mas você precisa ser capaz de reconhecer quando está sendo usado. Escolher senhas fortes também é importantíssimo para manter sua segurança, e por isso a melhor ideia é delegar essa responsabilidade a um gerenciador de senhas e permitir que o software crie senhas aleatórias para qualquer site em que você se logar.

Outra maneira de proteger suas comunicações é assinar e criptografar suas pastas de arquivos e emails com o GnuPG. dm-crypt e EncFS são duas alternativas para criptografar discos inteiros ou partições; eles usam, respectivamente, métodos de criptografia de bloco e empilhada.

Finalmente, o LibreOffice Impress é uma alternativa open source completa ao Microsoft Powerpoint, mas também existe o Beamer e o RevealJS para quem prefere criar apresentações usando código ao invés de interfaces gráficas. O ProjectLibre e o GanttProject podem ser boas escolhas para quem busca um substituto ao Microsoft Project.

Respostas aos Exercícios Guiados

1. Você deve usar uma “janela anônima” em seu navegador quando quer:

Navegar na internet de maneira completamente anônima	
Não deixar traços no computador que está usando	X
Ativar o TLS para evitar o rastreamento de cookies	
Para usar o DNT	
Para usar criptografia durante a transmissão de dados	

2. O que é OpenStack?

Um projeto que permite a criação de um IaaS privado	X
Um projeto que permite a criação de um PaaS privado	
Um projeto que permite a criação de um SaaS privado	
Um hipervisor	
Um gerenciador de senhas de código aberto	

3. Quais das opções abaixo são softwares válidos para criptografia de discos?

RevealJS, EncFS e dm-crypt	
dm-crypt e KeePass	
EncFS e Bitwarden	
EncFS e dm-crypt	X
TLS e dm-crypt	

4. Selecione verdadeiro ou falso para a criptografia de dispositivos com dm-crypt:

Os arquivos são criptografados antes de serem escritos no disco	V
O sistema de arquivos inteiro se torna uma massa criptografada	V
Apenas os arquivos e diretórios são criptografados, e não os links simbólicos	F
Não requer acesso de root	F
É uma criptografia de dispositivo de bloco	V

5. O Beamer é:

Um mecanismo de criptografia	
Um hipervisor	
Um programa de virtualização	
Um componente do OpenStack	
Uma ferramenta de apresentações em LaTeX	X

Respostas aos Exercícios Exploratórios

1. A maioria das distribuições vem com o Firefox instalado por padrão (se não for o seu caso, será preciso instalá-lo antes). Vamos instalar uma extensão para o Firefox chamada *Lightbeam*. Para isso, você pode pressionar `Ctrl` + `Shift` + `A` e buscar por "Lightbeam" no campo de busca que aparece na aba aberta, ou visitar a página da extensão com o Firefox e clicar no botão "Instalar": https://addons.mozilla.org/en-GB/firefox/addon/lightbeam-3-0/. Em seguida, inicie a extensão clicando no ícone e comece a visitar páginas web em outras abas para ver o que acontece.

   Lembra dos cookies que podem compartilhar seus dados com diferentes serviços quando você visita um website? É exatamente isso o que esta extensão vai lhe mostrar. O Lightbeam é uma experiência da Mozilla para tentar revelar os sites principais e de terceiros com os quais você interage ao visitar uma única URL. Esse conteúdo normalmente não está visível para o usuário médio e mostra que, às vezes, um único website interage com mais de uma dúzia de serviços.

2. Qual a coisa mais importante ao se usar um gerenciador de senhas?

   Ao usar um gerenciador de senhas, o mais importante a se ter em mente é memorizar sua senha mestre e usar uma senha aleatória única para cada serviço.

3. Use seu navegador para visitar https://haveibeenpwned.com/. Descubra qual a finalidade do site e confira se seu endereço de email foi incluído em algum vazamento de dados.

   O site mantém um banco de dados de informações de login cujas senhas foram afetadas por um vazamento de senhas. Ele permite buscar por um endereço de email e mostra se esse endereço estava incluído em um banco de dados público de credenciais roubadas. Há grandes chances de que seu endereço de email tenha sido afetado por um vazamento ou outro. Se for o caso, é melhor que suas senhas tenham sido atualizadas recentemente. Se você ainda não usa um gerenciador de senhas, dê uma olhada nas opções que recomendamos nesta lição.

Linux Professional Institute

Tópico 2: Encontrando seu caminho em um Sistema Linux

2.1 O básico sobre a linha de comando

Referência ao LPI objectivo

Linux Essentials version 1.6, Exam 010, Objective 2.1

Peso

3

Áreas chave de conhecimento

- Shell básico
- Sintaxe da linha de comando
- Variáveis
- Globbing (Englobamento)
- Uso de aspas

Segue uma lista parcial de arquivos, termos e utilitários utilizados

- Bash
- `echo`
- `history`
- Variável de ambiente `PATH`
- `export`
- `type`

2.1 Lição 1

Certificação:	Linux Essentials
Versão:	1.6
Tópico:	2 Como se orientar em um sistema Linux
Objetivo:	2.1 As bases da linha de comando
Lição:	1 de 2

Introdução

As distribuições Linux modernas oferecem uma ampla variedade de interfaces gráficas de usuário, mas um administrador sempre precisará saber como trabalhar com a linha de comando, ou *shell*. O shell é um programa que permite a comunicação por texto entre o sistema operacional e o usuário. Trata-se normalmente de um programa em modo texto que lê os dados inseridos pelo usuário e os interpreta na forma de comandos para o sistema.

Existem vários shells diferentes no Linux; estes são apenas alguns exemplos:

- Bourne-again shell (Bash)
- C shell (csh ou tcsh, a versão aprimorada do csh)
- Korn shell (ksh)
- Z shell (zsh)

No Linux, o mais comum é o shell Bash. É o que será usado em nossos exemplos e exercícios.

Ao usar um shell interativo, o usuário insere comandos no chamado prompt. Para cada distribuição Linux, o prompt padrão tem uma aparência um pouco diferente, mas ele geralmente

segue esta estrutura:

```
username@hostname diretório_atual tipo_de_shell
```

No Ubuntu ou Debian GNU/Linux, o prompt de um usuário comum provavelmente será assim:

```
carol@mycomputer:~$
```

O prompt do superusuário terá a seguinte aparência:

```
root@mycomputer:~#
```

No CentOS ou Red Hat Linux, o prompt para um usuário comum será semelhante a este:

```
[dave@mycomputer ~]$
```

E o prompt do superusuário será assim:

```
[root@mycomputer ~]#
```

Vamos entender melhor cada componente da estrutura:

`username`

Nome do usuário que está rodando o shell

`hostname`

Nome da máquina hospedeira na qual o shell é executado. Também existe um comando `hostname`, que permite exibir ou definir o nome de hospedeiro do sistema.

`current_directory`

O diretório em que o shell está atualmente. Um `~` indica que o shell está no diretório Home do usuário atual.

`shell_type`

`$` indica que o shell está sendo executado por um usuário comum.

`#` indica que o shell está sendo executado pelo superusuário `root`.

Como não precisamos de privilégios especiais, usaremos um prompt sem privilégios nos exemplos a seguir. Por questões de concisão, usaremos apenas o `$` como prompt.

Estrutura da linha de comando

A maioria dos comandos na linha de comando segue a mesma estrutura básica:

```
comando [opção(ões)/parâmetro(s)...] [argumento(s)...]
```

Veja, por exemplo, o seguinte comando:

```
$ ls -l /home
```

Vamos entender a finalidade de cada componente:

Comando

Programa que o usuário pretende executar — `ls`, no exemplo acima.

Opção(ões)/Parâmetro(s)

Um "botão" que modifica o comportamento do comando de alguma forma, como `-l` no exemplo acima. As opções podem ser acessadas na forma curta ou longa. Por exemplo, `-l` é idêntico a `--format = long`.

Também é possível combinar múltiplas opções e, no caso do formato abreviado, as letras geralmente podem ser digitadas juntas. Por exemplo, os seguintes comandos fazem todos a mesma coisa:

```
$ ls -al
$ ls -a -l
$ ls --all --format=long
```

Argumento(s)

Dados adicionais exigidos pelo programa, como um nome de arquivo ou caminho, como `/home` no exemplo acima.

A única parte obrigatória dessa estrutura é o próprio comando. Em geral, todos os outros elementos são opcionais, mas um programa pode exigir que determinadas opções, parâmetros ou argumentos sejam especificados.

NOTE A maioria dos comandos exibe uma visão geral dos comandos disponíveis quando executados com o parâmetro `--help`. Em breve, veremos maneiras adicionais de saber mais sobre os comandos do Linux.

Tipos de comportamento dos comandos

O shell aceita dois tipos de comandos:

Internos

Comandos que fazem parte do próprio shell e não são programas separados. Existem cerca de 30 desses comandos. Seu principal objetivo é executar tarefas dentro do shell (por exemplo, `cd`, `set`, `export`).

Externos

Comandos que residem em arquivos individuais. Esses arquivos geralmente são programas binários ou scripts. Quando um comando externo ao shell é executado, o shell usa a variável `PATH` para procurar um arquivo executável com o mesmo nome que o comando. Além dos programas instalados com o gerenciador de pacotes da distribuição, os usuários também podem criar seus próprios comandos externos.

O comando `type` mostra a que tipo pertence um comando específico:

```
$ type echo
echo is a shell builtin
$ type man
man is /usr/bin/man
```

Citação

Como usuário de Linux, você precisa saber criar ou manipular arquivos ou variáveis de diversas maneiras. Isso é fácil quando se trabalha com nomes de arquivos curtos e valores simples, mas fica mais complicado quando, por exemplo, precisamos usar espaços, caracteres especiais e variáveis. Os shells oferecem um recurso chamado citação, que encapsula esses dados usando diversos tipos de aspas (" ", ' '). No Bash, há três tipos de citações:

- Aspas duplas
- Aspas simples
- Caracteres de escape

Por exemplo, os seguintes comandos não agem da mesma maneira devido à citação:

```
$ TWOWORDS="two words"
$ touch $TWOWORDS
$ ls -l
-rw-r--r-- 1 carol carol     0 Mar 10 14:56 two
-rw-r--r-- 1 carol carol     0 Mar 10 14:56 words
$ touch "$TWOWORDS"
$ ls -l
-rw-r--r-- 1 carol carol     0 Mar 10 14:56  two
-rw-r--r-- 1 carol carol     0 Mar 10 14:58 'two words'
-rw-r--r-- 1 carol carol     0 Mar 10 14:56  words
$ touch '$TWOWORDS'
$ ls -l
-rw-r--r-- 1 carol carol     0 Mar 10 15:00 '$TWOWORDS'
-rw-r--r-- 1 carol carol     0 Mar 10 14:56  two
-rw-r--r-- 1 carol carol     0 Mar 10 14:58 'two words'
-rw-r--r-- 1 carol carol     0 Mar 10 14:56  words
```

NOTE A linha com `TWOWORDS =` é uma variável Bash que nós mesmos criamos. Falaremos das variáveis mais tarde. Isso serve apenas para mostrar como as citações afetam a saída das variáveis.

Aspas duplas

As aspas duplas dizem ao shell para considerar o texto entre as aspas ("...") como caracteres regulares. Todos os caracteres especiais perdem o significado, exceto `$` (cifrão), `\` (barra invertida) e `` ` `` (crase). Isso significa que as variáveis, substituições de comando e funções aritméticas ainda podem ser usadas.

Por exemplo, a substituição da variável `$USER` não é afetada pelas aspas duplas:

```
$ echo I am $USER
I am tom
$ echo "I am $USER"
I am tom
```

Um caractere de espaço, por outro lado, perde seu significado como separador de argumentos:

```
$ touch new file
$ ls -l
```

```
-rw-rw-r-- 1 tom students 0 Oct 8 15:18 file
-rw-rw-r-- 1 tom students 0 Oct 8 15:18 new
$ touch "new file"
$ ls -l
-rw-rw-r-- 1 tom students 0 Oct 8 15:19 new file
```

Como explicado, no primeiro exemplo o comando `touch` cria dois arquivos individuais; o comando interpreta as duas seqüências como argumentos individuais. No segundo exemplo, o comando interpreta as duas seqüências como um argumento e, portanto, cria apenas um arquivo. No entanto, recomenda-se evitar o caractere de espaço nos nomes de arquivos. Em vez disso, é preferível usar um sublinhado (`_`) ou um ponto (`.`).

Aspas simples

As aspas simples não têm as exceções das aspas duplas. Elas revogam qualquer significado especial de cada caractere. Se retomarmos um dos primeiros exemplos acima:

```
$ echo I am $USER
I am tom
```

Ao aplicar aspas simples, obtemos um resultado diferente:

```
$ echo 'I am $USER'
I am $USER
```

O comando agora exibe a sequência exata, sem substituir a variável.

Caracteres de escape

Podemos usar *caracteres de escape* para remover os significados especiais dos caracteres do Bash. Voltando à variável de ambiente `$USER`:

```
$ echo $USER
carol
```

Vemos que, por padrão, o conteúdo da variável é exibido no terminal. No entanto, se colocarmos um caractere de barra invertida (`\`) antes do cifrão, o significado especial do cifrão será cancelado. Por sua vez, isso não permitirá que o Bash expanda o valor da variável para o nome de usuário da pessoa que está executando o comando, mas, em vez disso, interpretará o nome da variável

literalmente:

```
$ echo \$USER
$USER
```

Como você deve lembrar, é possível obter resultados semelhantes usando aspas simples, que imprimem o conteúdo literal do que estiver entre elas. No entanto, o funcionamento do caractere de escape é diferente, pois instrui o Bash a ignorar qualquer significado especial do caractere que ele precede.

Exercícios Guiados

1. Divida as linhas abaixo nos componentes de comando, opção(ões)/parâmetro(s) e argumento(s):
   - Exemplo: `cat -n /etc/passwd`

Comando:	`cat`
Opção:	`-n`
Argumento:	`/etc/passwd`

   - `ls -l /etc`

Comando:	
Opção:	
Argumento:	

   - `ls -l -a`

Comando:	
Opção:	
Argumento:	

   - `cd /home/user`

Comando:	
Opção:	
Argumento:	

2. Descubra de que tipo são os comandos a seguir:

   Exemplo:

`pwd`	Embutido no Shell
`mv`	Comando externo

`cd`	

`cat`	
`exit`	

3. Resolva os seguintes comandos que usam citações:

   Exemplo:

`echo "$HOME is my home directory"`	`echo /home/user is my home directory`

`touch "$USER"`	
`touch 'touch'`	

Exercícios Exploratórios

1. Com um só comando e usando a expansão de chaves no Bash (veja a página de manual do Bash), crie 5 arquivos numerados de 1 a 5 com o prefixo `game` (`game1`, `game2`, ...).

2. Remova os cinco arquivos que acaba de criar com um único comando, usando um caractere especial diferente (consulte *Pathname Expansion* nas páginas de manual do Bash).

3. Existem outros jeitos de fazer com que dois comandos interajam entre si? Quais?

Resumo

Nesta lição, você aprendeu:

- Conceitos do shell do Linux
- O que é o shell Bash
- A estrutura da linha de comando
- Uma introdução às citações

Comandos usados nos exercícios:

`bash`
O shell mais popular em computadores com Linux.

`echo`
Gera texto no terminal.

`ls`
Lista o conteúdo de um diretório.

`type`
Mostra como um comando específico é executado.

`touch`
Cria um arquivo vazio ou atualiza a data de modificação de um arquivo existente.

`hostname`
Mostra ou altera o nome do host de um sistema.

Respostas aos Exercícios Guiados

1. Divida as linhas abaixo nos componentes de comando, opção(ões)/parâmetro(s) e argumento(s):
   - `ls -l /etc`

Comando:	`ls`
Opção:	`-l`
Argumento:	`/etc`

   - `ls -l -a`

Comando:	`ls`
Opção:	`-l -a`
Argumento:	

   - `cd /home/user`

Comando:	`cd`
Opção:	
Argumento:	`/home/user`

2. Descubra de que tipo são os comandos a seguir:

`cd`	Embutido no shell
`cat`	Comando externo
`exit`	Embutido no shell

3. Resolva os seguintes comandos que usam citações:

`touch "$USER"`	`tom`
`touch 'touch'`	Cria um arquivo chamado `touch`

Respostas aos Exercícios Exploratórios

1. Com um só comando e usando a expansão de chaves no Bash (veja a página de manual do Bash), crie 5 arquivos numerados de 1 a 5 com o prefixo `game` (`game1`, `game2`, ...).

   É possível usar faixas para expressar os números de 1 a 5 dentro de um só comando:

   ```
   $ touch game{1..5}
   $ ls
   game1  game2  game3  game4  game5
   ```

2. Remova os cinco arquivos que acaba de criar com um único comando, usando um caractere especial diferente (consulte *Pathname Expansion* nas páginas de manual do Bash).

   Como todos os arquivos começam com `game` e terminam com um só caractere (neste caso, um número de 1 a 5), `?` pode ser usado como caractere especial no lugar do último caractere de um nome de arquivo:

   ```
   $ rm game?
   ```

3. Existem outros jeitos de fazer com que dois comandos interajam entre si? Quais?

   Sim, um comando pode, por exemplo, escrever dados em um arquivo que é em seguida processado por outro comando. O Linux também pode coletar a saída de um comando e usá-la como entrada de outro comando. Isso se chama *piping* e aprenderemos mais a respeito em uma lição futura.

2.1 Lição 2

Certificação:	Linux Essentials
Versão:	1.6
Tópico:	2 Como se orientar em um sistema Linux
Objetivo:	2.1 As bases da linha de comando
Lição:	2 de 2

Introdução

Todos os shells gerenciam um conjunto de informações de status ao longo das sessões do shell. Essas informações de tempo de execução podem mudar durante a sessão e influenciar o comportamento do shell. Esses dados também são usados pelos programas para determinar aspectos da configuração do sistema. A maioria desses dados é armazenada nas chamadas *variáveis*, que abordaremos nesta lição.

Variáveis

As variáveis são espaços de armazenamento para dados, como texto ou números. Uma vez definido, o valor de uma variável pode ser acessado posteriormente. As variáveis têm um nome, o que permite acessar uma variável específica mesmo quando seu conteúdo é alterado. Elas são uma ferramenta muito comum na maioria das linguagens de programação.

Na maioria dos shells do Linux, existem dois tipos de variáveis:

Variáveis locais

Essas variáveis estão disponíveis apenas para o processo atual do shell. Se você criar uma

variável local e, em seguida, iniciar outro programa nesse shell, a variável não poderá mais ser acessada por esse programa. Como não são herdadas por subprocessos, essas variáveis são chamadas *variáveis locais*.

Variáveis de ambiente

Essas variáveis estão disponíveis tanto em uma sessão de shell específica quanto em subprocessos gerados a partir dessa sessão de shell. Essas variáveis podem ser usadas para transmitir dados de configuração para os comandos executados. Como os programas podem acessar essas variáveis, elas são chamadas *variáveis de ambiente*. A maioria das variáveis de ambiente aparece em letras maiúsculas (por exemplo, `PATH`, `DATE`, `USER`). Um conjunto de variáveis de ambiente padrão fornece, por exemplo, informações sobre o diretório inicial ou o tipo de terminal do usuário. Em certos casos, nos referimos ao conjunto completo de todas as variáveis de ambiente como *ambiente*.

NOTE As variáveis não são persistentes. Quando o shell em que foram configuradas é fechado, todas as variáveis e seus conteúdos são perdidos. A maioria dos shells oferece arquivos de configuração que contêm variáveis definidas sempre que um novo shell é iniciado. As variáveis que devem ser definidas permanentemente precisam ser adicionadas a um desses arquivos de configuração.

Manipulação de variáveis

Como administrador do sistema, você precisará criar, modificar ou remover variáveis locais e de ambiente.

Trabalhando com variáveis locais

Você pode configurar uma variável local usando o operador `=` (igual). Uma atribuição simples criará uma variável local:

```
$ greeting=hello
```

NOTE Não coloque espaço antes ou depois do operador `=`.

É possível exibir qualquer variável usando o comando `echo`. O comando geralmente exibe o texto na seção de argumentos:

```
$ echo greeting
greeting
```

Para acessar o valor da variável, você terá de usar `$` (cifrão) na frente do nome da variável.

```
$ echo $greeting
hello
```

Como vimos, a variável foi criada. Agora abra outro shell e tente exibir o conteúdo da variável criada.

```
$ echo $greeting
```

Nada é exibido. Isso ilustra o fato de que as variáveis existem somente em um shell específico.

Para verificar se a variável é de fato uma variável local, tente gerar um novo processo e confira se esse processo é capaz de acessar a variável. Para isso, inicie outro shell e execute nele o comando `echo`. Como o novo shell roda em um novo processo, ele não herdará as variáveis locais do processo pai:

```
$ echo $greeting world
hello world
$ bash -c 'echo $greeting world'
world
```

NOTE Lembre-se de usar aspas simples no exemplo acima.

Para remover uma variável, usamos o comando `unset`:

```
$ echo $greeting
hey
$ unset greeting
$ echo $greeting
```

NOTE O `unset` requer o nome da variável como argumento. Portanto, não é possível adicionar `$` ao nome, já que isso resolveria a variável e passaria o valor da variável para `unset` em vez do nome da variável.

Trabalhando com variáveis globais

Para disponibilizar uma variável local para subprocessos, podemos transformá-la em variável de ambiente. Usamos para isso o comando `export`. Quando ele é invocado junto ao nome da

variável, essa variável é adicionada ao ambiente do shell:

```
$ greeting=hello
$ export greeting
```

NOTE Aqui também não se deve usar `$` ao executar `export`, já que queremos transmitir o nome da variável e não seu conteúdo.

Uma maneira mais fácil de criar a variável de ambiente é combinar os dois métodos acima, atribuindo o valor da variável na parte de argumento do comando.

```
$ export greeting=hey
```

Vamos conferir novamente se a variável está acessível aos subprocessos:

```
$ export greeting=hey
$ echo $greeting world
hey world
$ bash -c 'echo $greeting world'
hey world
```

Outra maneira de usar variáveis de ambiente é colocá-las na frente dos comandos. Vamos testar essa possibilidade com a variável de ambiente `TZ`, que contém o fuso horário. Essa variável é usada pelo comando `date` para determinar qual hora do fuso horário deve ser exibida:

```
$ TZ=EST date
Thu 31 Jan 10:07:35 EST 2019
$ TZ=GMT date
Thu 31 Jan 15:07:35 GMT 2019
```

Para exibir todas as variáveis de ambiente, use o comando `env`.

A variável `PATH`

A variável `PATH` é uma das variáveis de ambiente mais importantes de um sistema Linux. Ela armazena uma lista de diretórios, separados por dois pontos, que contêm programas executáveis que funcionam como comandos do shell do Linux.

```
$ echo $PATH
```

```
/home/user/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
```

Para acrescentar um novo diretório à variável, usamos o sinal de dois pontos (`:`).

```
$ PATH=$PATH:new_directory
```

Eis um exemplo:

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
$ PATH=$PATH:/home/user/bin
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/home/user/bin
```

Como vimos, `$PATH` é usado no novo valor atribuído a `PATH`. Essa variável é resolvida durante a execução do comando e garante que o conteúdo original da variável seja preservado. Obviamente, também dá para usar outras variáveis na atribuição:

```
$ mybin=/opt/bin
$ PATH=$PATH:$mybin
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/home/user/bin:/opt/bin
```

A variável `PATH` deve ser usada com cautela, pois é crucial para o trabalho na linha de comando. Vamos considerar a seguinte variável `PATH`:

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

Para descobrir como o shell chama um comando específico, `which` pode ser executado com o nome do comando como argumento. Podemos, por exemplo, tentar descobrir onde o `nano` está armazenado:

```
$ which nano
/usr/bin/nano
```

Neste exemplo, o executável `nano` está localizado no diretório `/usr/bin`. Vamos remover o diretório da variável e verificar se o comando ainda funciona:

```
$ PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/sbin:/bin:/usr/games
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/sbin:/bin:/usr/games
```

Vamos procurar pelo comando `nano` novamente:

```
$ which nano
which: no nano in (/usr/local/sbin:/usr/local/bin:/usr/sbin:/sbin:/bin:/usr/games)
```

Como vimos, o comando não foi encontrado, e portanto não foi executado. A mensagem de erro também explica o motivo pelo qual o comando não foi encontrado e em quais locais foi buscado.

Vamos readicionar os diretórios e tentar executar o comando novamente.

```
$ PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
$ which nano
/usr/bin/nano
```

Agora nosso comando voltou a funcionar.

TIP A ordem dos elementos em `PATH` também define a ordem de pesquisa. O primeiro executável da lista que corresponda às especificações será executado.

Exercícios Guiados

1. Crie uma variável local `number`.

2. Crie uma variável de ambiente `ORDER` usando um dos dois métodos acima.

3. Exiba o nome das variáveis e seu conteúdo.

4. Qual o escopo das variáveis criadas anteriormente?

Exercícios Exploratórios

1. Crie uma variável local `nr_files` e atribua o número de linhas encontrado no arquivo `/etc/passwd`. Dica: pesquise sobre o comando `wc` e substituição de comandos, e não se esqueça das aspas.

2. Crie uma variável de ambiente `ME`. Atribua a ela o valor de variável `USER`.

3. Inclua o valor da variável `HOME` a `ME`, usando o delimitador `:`. Exiba o conteúdo da variável `ME`.

4. Usando o exemplo acima, crie uma variável chamada `today` e atribua a data de um dos fusos horários.

5. Crie outra variável chamada `today1` e atribua a ela a data do sistema.

Sumário

Nesta lição, você aprendeu:

- Tipos de variáveis
- Como criar variáveis
- Como manipular variáveis

Comandos usados nos exercícios:

`env`
: Exibe o ambiente atual.

`echo`
: Gera uma saída de texto.

`export`
: Torna as variáveis locais disponíveis para os subprocessos.

`unset`
: Remove uma variável.

Respostas aos Exercícios Guiados

1. Crie uma variável local `number`.

```
$ number=5
```

2. Crie uma variável de ambiente `ORDER` usando um dos dois métodos acima.

```
$ export ORDER=desc
```

3. Exiba o nome das variáveis e seu conteúdo.

```
$ echo number
number
$ echo ORDER
ORDER
$ echo $number
5
$ echo $ORDER
desc
```

4. Qual o escopo das variáveis criadas anteriormente?
   - O escopo da variável `number` é somente o shell atual.
   - O escopo da variável `ORDER` é o shell atual e todos os subshells gerados por ele.

Respostas aos Exercícios Exploratórios

1. Crie uma variável local `nr_files` e atribua o número de linhas encontrado no arquivo `/etc/passwd`. Dica: pesquise sobre o comando `wc` e substituição de comandos, e não se esqueça das aspas.

```
$ nr_files=`wc -l /etc/passwd`
```

2. Crie uma variável de ambiente `ME`. Atribua a ela o valor de variável `USER`.

```
$ export ME=$USER
```

3. Inclua o valor da variável `HOME` a `ME`, usando o delimitador `:`. Exiba o conteúdo da variável `ME`.

```
$ ME=$ME:$HOME
$ echo $ME
user:/home/user
```

4. Usando o exemplo acima, crie uma variável chamada `today` e atribua a data de um dos fusos horários.

   O exemplo a seguir usa os fusos horários GMT e EST, mas você pode escolher o que quiser.

```
$ today=$(TZ=GMT date)
$ echo $today
Thu 31 Jan 15:07:35 GMT 2019
```

   or

```
$ today=$(TZ=EST date)
$ echo $today
Thu 31 Jan 10:07:35 EST 2019
```

5. Crie outra variável chamada `today1` e atribua a ela a data do sistema.

   Se você estiver no horário GMT:

```
$ today1=$(date)
```

```
$ echo today1
Thu 31 Jan 10:07:35 EST 2019
```

2.2 O Usando a linha de comando para conseguir ajuda

Referência ao LPI objectivo

Linux Essentials version 1.6, Exam 010, Objective 2.2

Peso

2

Áreas chave de conhecimento

- Man
- Info

Segue uma lista parcial de arquivos, termos e utilitários utilizados

- `man`
- `info`
- `/usr/share/doc/`
- `locate`

2.2 Lição 1

Certificação:	Linux Essentials
Versão:	1.6
Tópico:	2 Como se orientar em um sistema Linux
Objetivo:	2.2 Como obter ajuda na linha de comando
Lição:	1 de 1

Introdução

A linha de comando é uma ferramenta bastante complexa. Cada comando possui suas próprias opções únicas; por isso, é essencial conhecer a documentação ao se trabalhar com um sistema Linux. Além do diretório `/usr/share/doc/`, que armazena a maior parte da documentação, diversas outras ferramentas fornecem informações sobre o uso dos comandos do Linux. Este capítulo se concentra em métodos para acessar essa documentação, com o objetivo de obter ajuda.

Existem vários métodos para obter ajuda na linha de comando do Linux. Alguns exemplos são `man`, `help` e `info`. Para o Linux Essentials, focaremos em `man` e `info`, que são as ferramentas mais usadas para obter ajuda.

Outro tópico deste capítulo será a localização de arquivos. Trabalharemos principalmente com o comando `locate`.

Como obter ajuda na linha de comando

Ajuda interna

Quando são iniciados com o parâmetro `--help`, a maioria dos comandos exibe algumas breves instruções sobre seu uso. Embora nem todos os comandos ofereçam essa opção, ainda vale a pena experimentá-la na primeira vez que se usa o comando para aprender mais sobre os parâmetros dele. Esteja ciente de que as instruções de `--help` geralmente são bastante concisas em comparação com outras fontes de documentação que discutiremos nesta lição.

Páginas man

A maioria dos comandos inclui uma página de manual, ou página "man". Essa documentação geralmente é instalada com o software e pode ser acessada com o comando `man`. O comando cuja página de manual deve ser exibida é adicionado a `man` na forma de argumento:

```
$ man mkdir
```

Esse comando abre a página man de `mkdir`. Use as setas direcionais ou a barra de espaço para navegar pela página do manual. Para sair da página man, pressione `Q`.

Cada página man é dividida em no máximo 11 seções, embora muitas delas sejam opcionais:

Seção	**Descrição**
NAME	Nome do comando e breve descrição
SYNOPSIS	Descrição da sintaxe do comando
DESCRIPTION	Descrição dos efeitos do comando
OPTIONS	Opções disponíveis
ARGUMENTS	Argumentos disponíveis
FILES	Arquivos auxiliares
EXAMPLES	Uma amostra da linha de comando
SEE ALSO	Referências cruzadas aos tópicos relacionados
DIAGNOSTICS	Mensagens de Advertência e Erro
COPYRIGHT	Autor(es) do comando
BUGS	Limitações conhecidas do comando

Na prática, a maioria das páginas de manual não contém todas essas partes.

As páginas man estão organizadas em oito categorias, numeradas de 1 a 8:

Categoria	**Descrição**
1	Comando do usuário
2	Chamadas do sistema
3	Funções da biblioteca C
4	Drivers e arquivos de dispositivo
5	Arquivos de configuração e formatos de arquivo
6	Jogos
7	Miscelânea
8	Comandos do administrador do sistema
9	Funções do kernel (não padrão)

Cada página de manual pertence a exatamente uma seção. No entanto, seções diferentes podem conter páginas de manual com o mesmo nome. Vejamos, por exemplo, o comando `passwd`. Esse comando pode ser usado para alterar a senha de um usuário. Como `passwd` é um comando do usuário, sua página de manual reside na seção 1. Além do comando `passwd`, o arquivo de banco de dados de senhas `/etc/passwd` também possui uma página de manual chamada `passwd`. Como neste caso se trata de um arquivo de configuração, ele pertence à seção 5. Nas referências a uma página de manual, a categoria é frequentemente adicionada ao nome da página, por exemplo `passwd(1)` ou `passwd(5)`, para identificar a página respectiva.

Por padrão, `man passwd` exibe a primeira página man disponível; neste caso, `passwd(1)`. A categoria da página man desejada pode ser especificada em um comando como `man 1 passwd` ou `man 5 passwd`.

Já explicamos como navegar por uma página man e como retornar à linha de comando. Internamente, o `man` usa o comando `less` para exibir o conteúdo da página. `less` permite procurar texto dentro de uma página man. Para procurar a palavra `linux`, podemos simplesmente digitar `/linux` para pesquisar a partir do ponto em que estamos na página, ou `?Linux` para iniciar uma pesquisa reversa. Esta ação destaca todos os resultados correspondentes e move a página para a primeira correspondência destacada. Nos dois casos, basta digitar `N` para pular até a correspondência seguinte. Para encontrar mais informações sobre esses recursos adicionais, pressione `H`; será exibido um menu com todas as informações.

Páginas info

Outra ferramenta muito útil ao trabalhar com o sistema Linux são as páginas de informações, ou info. As páginas info geralmente são mais detalhadas que as páginas man e são formatadas em hipertexto, semelhantes às páginas web na internet.

Para exibir as páginas info, usamos:

```
$ info mkdir
```

Nessas páginas, o comando `info` lê um arquivo de informações estruturado em nós individuais dentro de uma árvore. Cada nó contém um tópico simples e o comando `info` inclui hiperlinks que ajudam a passar de um para o outro. Para acessar o link, pressione `Enter`, com o cursor pousado sobre um dos asteriscos principais.

A exemplo de `man`, a ferramenta `info` também inclui comandos de navegação na página. Para descobrir mais sobre esses comandos, pressione `?` enquanto estiver na página de informações. Essas ferramentas o ajudarão a navegar na página mais facilmente e a entender como acessar os nós e mover-se dentro da árvore de nós.

O diretório `/usr/share/doc/`

Como já mencionado, o diretório `/usr/share/doc/` armazena a maior parte da documentação dos comandos que estão em uso. Essa pasta contém um diretório para a maioria dos pacotes instalados no sistema. O nome do diretório geralmente é o nome do pacote e, ocasionalmente, a versão. Esses diretórios incluem um arquivo `README` ou `readme.txt` contendo a documentação básica do pacote. Juntamente com o arquivo `README`, a pasta também pode conter outros arquivos de documentação, como o registro de alterações (changelog), que inclui o histórico do programa em detalhes, ou exemplos de arquivos de configuração para aquele pacote específico.

As informações contidas no arquivo `README` variam de um pacote a outro. Todos os arquivos são gravados em texto simples e, portanto, podem ser lidos em qualquer editor de texto de sua preferência. O número exato e os tipos de arquivos dependem do pacote. Verifique alguns dos diretórios para obter uma visão geral de seu conteúdo.

Como localizar arquivos

O comando `locate`

Um sistema Linux é construído a partir de diversos diretórios e arquivos. O Linux tem muitas

ferramentas que permitem localizar um arquivo específico em um sistema. A mais rápida é o comando `locate`.

O `locate` pesquisa dentro de um banco de dados e, em seguida, gera todos os nomes que correspondem à cadeia de caracteres (string) especificada:

```
$ locate note
/lib/udev/keymaps/zepto-znote
/usr/bin/zipnote
/usr/share/doc/initramfs-tools/maintainer-notes.html
/usr/share/man/man1/zipnote.1.gz
```

O comando `locate` também suporta o uso de curingas e expressões regulares; assim, o termo de pesquisa não precisa corresponder ao nome inteiro do arquivo desejado. Trataremos mais a fundo das expressões regulares em um capítulo posterior.

Por padrão, `locate` se comporta como se o termo de busca estivesse cercado por asteriscos. Portanto, `locate TERMO` é o mesmo que `locate *TERMO*`. Isso permite fornecer apenas substrings em vez do nome exato do arquivo. Se quiser modificar esse comportamento, use as diferentes opções especificadas na página man de `locate`.

Como `locate` consulta um banco de dados, você pode não encontrar um arquivo que foi criado recentemente. O banco de dados é gerenciado por um programa chamado `updatedb`. Ele costuma ser executado periodicamente, mas se você possui privilégios de root e precisa atualizar o banco de dados imediatamente, pode executar o comando `updatedb` a qualquer momento.

O comando `find`

O `find` é outra ferramenta muito popular usada para procurar arquivos. Esse comando tem uma abordagem diferente do `locate`. O comando `find` pesquisa recursivamente em uma árvore de diretórios, incluindo os subdiretórios. O `find` faz essa pesquisa a cada chamada, em vez de manter um banco de dados como o `locate`. A exemplo do `locate`, o `find` também suporta caracteres curinga e expressões regulares.

O `find` requer pelo menos o local em que ele deve procurar. Além disso, podemos adicionar expressões para filtrar os arquivos a exibir. Um exemplo é a expressão `-name`, que procura por arquivos com um nome específico:

```
~$ cd Downloads
~/Downloads
$ find . -name thesis.pdf
```

```
./thesis.pdf
~/Downloads
$ find ~ -name thesis.pdf
/home/carol/Downloads/thesis.pdf
```

O primeiro comando `find` procura por arquivos dentro do atual diretório `Downloads`, ao passo que o segundo busca pelo arquivo no diretório Home do usuário.

O comando `find` é muito complexo e, portanto, não será tratado no exame Linux Essentials. No entanto, trata-se de uma ferramenta poderosa e particularmente útil no cotidiano.

Exercícios Guiados

1. Use o comando `man` para descobrir o que cada comando faz:

Comando	**Descrição**
`ls`	Exibe o conteúdo de um diretório.
`cat`	
`cut`	
`cd`	
`cp`	
`mv`	
`mkdir`	
`touch`	
`wc`	
`passwd`	
`rm`	
`rmdir`	
`more`	
`less`	
`whereis`	
`head`	
`tail`	
`sort`	
`tr`	
`chmod`	
`grep`	

2. Abra a página info de `ls` e identifique o MENU.
   - Quais são as opções?
   - Encontre a opção que permite classificar os dados de saída por data de modificação.

3. Exiba o caminho para os três primeiros arquivos `README`. Use o comando `man` para identificar a opção correta para `locate`.

4. Crie um arquivo chamado `test` em seu diretório Home. Encontre seu caminho absoluto usando o comando `locate`.

5. Você o encontrou imediatamente? O que teve de fazer para que `locate` o encontrasse?

6. Procure pelo arquivo de teste criado anteriormente usando o comando `find`. Qual sintaxe você usou e qual o caminho absoluto?

Exercícios Exploratórios

1. Um dos comandos na tabela acima não tem uma página `man`. Qual é esse comando e por que você acha que ele não tem uma página man?

2. Usando os comandos da tabela acima, crie a seguinte árvore de arquivos. Os nomes que começam com letra maiúscula são diretórios e os que começam em minúscula são arquivos.

```
User
├── Documents
|   ├──Hello
|   |   ├──hey2
|   |   ├──helloa
|   |   └──ola5
|   └──World
|       └──earth9
├── Downloads
|   ├──Music
|   └──Songs
|       ├──collection1
|       └──collection2
├── Test
|   └── passa
└── test
```

3. Exiba na tela o diretório de trabalho atual, incluindo as subpastas.

4. Procure dentro da árvore por todos os arquivos terminados com um número.

5. Remova a árvore de diretórios inteira com um único comando.

Resumo

Nesta lição, você aprendeu:

- Como conseguir ajuda
- Como usar o comando `man`
- Como navegar na página `man`
- Diferentes seções da página `man`
- Como usar o comando `info`
- Como navegar entre diferentes nós
- Como buscar arquivos dentro do sistema

Comandos usados nos exercícios:

`man`
: Exibe uma página de manual.

`info`
: Exibe uma página de informações.

`locate`
: Busca no banco de dados de `locate` por arquivos com um nome específico.

`find`
: Busca no sistema de arquivos por nomes que correspondam a uma série de critérios de seleção.

`updatedb`
: Atualiza o banco de dados de `locate`.

Respostas aos Exercícios Guiados

1. Use o comando `man` para descobrir o que cada comando faz:

Comando	**Descrição**
`ls`	Exibe o conteúdo de um diretório.
`cat`	Concatena ou visualiza arquivos de texto
`cut`	Remove seções de um arquivo de texto
`cd`	Passa para um diretório diferente
`cp`	Copia um arquivo
`mv`	Move um arquivo (também pode ser usado para renomear)
`mkdir`	Cria um novo diretório
`touch`	Cria um arquivo ou modifica a hora e data da última modificação de um arquivo existente
`wc`	Conta o número de palavras, linhas ou bytes de um arquivo
`passwd`	Muda a senha de um usuário
`rm`	Remove um arquivo
`rmdir`	Remove um diretório
`more`	Visualiza arquivos de texto uma tela de cada vez
`less`	Visualiza arquivos de texto, permite rolar uma linha ou página por vez, para cima ou para baixo
`whereis`	Exibe o caminho até um programa especificado e arquivos de manual relacionados
`head`	Exibe as primeiras linhas de um arquivo
`tail`	Exibe as últimas linhas de um arquivo
`sort`	Ordena um arquivo numérica ou alfabeticamente

Comando	Descrição
`tr`	Traduz ou remove caracteres de um arquivo
`chmod`	Altera as permissões de um arquivo
`grep`	Busca dentro de um arquivo

2. Abra a página info de `ls` e identifique o MENU.*
   - Quais são as opções?
     - Quais arquivos são listados
     - Qual informação é listada
     - Classificação do resultado
     - Informações sobre a versão
     - Formatação geral da saída
     - Formatação dos registros de data e hora do arquivo
     - Formatação dos nomes dos arquivos
   - Encontre a opção que permite classificar os dados de saída por data de modificação.

     `-t` ou `--sort=time`

3. Exiba o caminho para os três primeiros arquivos `README`. Use o comando `man` para identificar a opção correta para `locate`.

```
$ locate -l 3 README
/etc/alternatives/README
/etc/init.d/README
/etc/rc0.d/README
```

4. Crie um arquivo chamado `test` em seu diretório Home. Encontre seu caminho absoluto usando o comando `locate`.

```
$ touch test
$ locate test
/home/user/test
```

5. Você o encontrou imediatamente? O que teve de fazer para que `locate` o encontrasse?

```
$ sudo updatedb
```

O arquivo acaba de ser criado, portanto ainda não foi registrado no banco de dados.

6. Procure pelo arquivo de teste criado anteriormente usando o comando `find`. Qual sintaxe você usou e qual o caminho absoluto?

```
$ find ~ -name test
```

ou

```
$ find . -name test
/home/user/test
```

Respostas aos Exercícios Exploratórios

1. Um dos comandos na tabela acima não tem uma página `man`. Qual é esse comando e por que você acha que ele não tem uma página man?

   O comando `cd`. Ele não tem uma página man porque é um comando interno do shell.

2. Usando os comandos da tabela acima, crie a seguinte árvore de arquivos. Os nomes que começam com letra maiúscula são diretórios e os que começam em minúscula são arquivos.

```
User
├── Documents
|   ├──Hello
|   |   ├──hey2
|   |   ├──helloa
|   |   └──ola5
|   └──World
|       └──earth9
├── Downloads
|   ├──Music
|   └──Songs
|       ├──collection1
|       └──collection2
├── Test
|   └── passa
└── test
```

   A solução é uma combinação dos comandos `mkdir` e `touch`.

3. Exiba na tela o diretório de trabalho atual, incluindo as subpastas.

```
$ ls -R
```

4. Procure dentro da árvore por todos os arquivos terminados com um número.

```
$ find ~ -name "*[0-9]"
$ locate "*[0-9]"
```

5. Remova a árvore de diretórios inteira com um único comando.

```
$ rm -r Documents Downloads Test test
```

2.3 Usando diretórios e listando arquivos

Referência ao LPI objectivo

Linux Essentials version 1.6, Exam 010, Objective 2.3

Peso

2

Áreas chave de conhecimento

- Arquivos, diretórios
- Arquivos e diretórios ocultos
- Home
- Caminhos relativos e absolutos

Segue uma lista parcial de arquivos, termos e utilitários utilizados

- Opções comuns para `ls`
- Listagens recursivas
- `cd`
- `.` e `..`
- `home` e `~`

Linux Professional Institute

2.3 Lição 1

Introdução

Certificação:	Linux Essentials
Versão:	1.6
Tópico:	2 Como se orientar em um sistema Linux
Objetivo:	2.3 Como usar diretórios e listar arquivos
Lição:	1 de 2

Arquivos e diretórios

O sistema de arquivos do Linux é semelhante ao sistema de arquivos de outros sistemas operacionais, pois contém *arquivos* e *diretórios*. Os arquivos contêm dados, como texto legível por humanos, programas executáveis ou dados binários usados pelo computador. Os diretórios são usados para organizar o sistema de arquivos. Eles podem conter arquivos e outros diretórios.

```
$ tree

Documents
├── Mission-Statement.txt
└── Reports
    └── report2018.txt

1 directory, 2 files
```

Neste exemplo, `Documents` é um diretório que contém um arquivo (`Mission-Statement.txt`) e um *subdiretório* (`Reports`). O diretório `Reports`, por sua vez, contém um arquivo chamado `report2018.txt`. O diretório `Documents` é considerado o *pai* do diretório `Reports`.

TIP Se o comando `tree` não estiver disponível no seu sistema, instale-o usando o gerenciador de pacotes da sua distribuição Linux. Consulte a lição sobre gerenciamento de pacotes para aprender como fazer isso.

Nomes de arquivos e diretórios

Os nomes de arquivos e diretórios no Linux podem conter letras minúsculas e maiúsculas, números, espaços e caracteres especiais. No entanto, como muitos caracteres especiais têm um significado preciso no shell do Linux, é recomendável não usar espaços ou caracteres especiais ao nomear arquivos ou diretórios. Para usar espaços, por exemplo, é preciso que o *caractere de escape* `\` seja inserido corretamente:

```
$ cd Mission\ Statements
```

Além disso, veja o nome do arquivo `report2018.txt`. Os nomes de arquivos podem conter um *sufixo* após o ponto final (`.`). Ao contrário do que acontece no Windows, esse sufixo não tem significado especial no Linux; ele existe para a compreensão humana. Em nosso exemplo, `.txt` indica para nós que este é um arquivo de texto sem formatação, embora, tecnicamente, ele possa conter qualquer tipo de dados.

Navegando no sistema de arquivos

Como obter a localização atual

Os shells do Linux, como o Bash, são baseados em texto, e por isso é importante conhecer sua localização atual ao navegar no sistema de arquivos. O *prompt de comando* fornece essas informações:

```
user@hostname ~/Documents/Reports $
```

Trataremos de informações como `user` e `hostname` futuramente. Graças ao prompt, sabemos que nossa localização atual é o diretório `Reports`. Da mesma forma, o comando `pwd` serve para *imprimir o diretório de trabalho*:

```
user@hostname ~/Documents/Reports $ pwd
```

```
/home/user/Documents/Reports
```

A relação entre os diretórios é representada com uma barra (`/`). Sabemos que `Reports` é um subdiretório de `Documents`, que por sua vez é um subdiretório de `user`, localizado em um diretório chamado `home`. O `home` não parece ter um diretório pai, mas isso não é verdade. O pai de `home` se chama *root* e é representado pela primeira barra (`/`). Explicaremos o diretório root em uma seção posterior.

Note que a saída do comando `pwd` difere um pouco do caminho mostrado no prompt de comando. Em vez de `/ home/user`, o prompt de comando contém um til (`~`). O til é um caractere especial que representa o diretório inicial do usuário. Trataremos disso em mais detalhes na próxima lição.

Como listar o conteúdo do diretório

O conteúdo do diretório atual é listado com o comando `ls`:

```
user@hostname ~/Documents/Reports $ ls
report2018.txt
```

Note que o `ls` não fornece informações sobre o diretório pai. Da mesma forma, por padrão, `ls` não exibe nenhuma informação sobre o conteúdo dos subdiretórios. O `ls` pode apenas "ver" o que está no diretório atual.

Como mudar de diretório

A navegação no Linux é feita principalmente com o comando `cd`. Esse comando *muda de diretório*. Usando o comando `pwd` como anteriormente, sabemos que nosso diretório atual é `/home/user/Documents/Reports`. Podemos mudar nosso diretório atual inserindo um novo caminho:

```
user@hostname ~ $ cd /home/user/Documents
user@hostname ~/Documents $ pwd
/home/user/Documents
user@hostname ~/Documents $ ls
Mission-Statement.txt Reports
```

Neste novo local, podemos "ver" `Mission-Statement.txt` e o subdiretório `Reports`, mas não o conteúdo de nosso subdiretório. Podemos navegar de volta para `Relatórios` assim:

```
user@hostname ~/Documents $ cd Reports
user@hostname ~/Documents/Reports $ pwd
/home/user/Documents/Reports
user@hostname ~/Documents/Reports $ ls
report2018.txt
```

Agora estamos de volta onde começamos.

Caminhos absolutos e relativos

O comando `pwd` sempre imprime um *caminho absoluto*, ou seja, o caminho contém todas as etapas, desde o início do sistema de arquivos (`/`) até o final (`Reports`). Os caminhos absolutos sempre começam com um `/`.

```
/
└── home
    └── user
        └── Documents
            └── Reports
```

O caminho absoluto contém todas as informações necessárias para se chegar a `Reports` a partir de qualquer lugar no sistema de arquivos. A desvantagem é que é tedioso digitá-lo inteiro.

O segundo exemplo (`cd Reports`) foi muito mais fácil de digitar. Esse é um exemplo de *caminho relativo*. Os caminhos relativos são mais curtos, mas só têm significado em relação à sua localização atual. Considere esta analogia: eu estou visitando você em sua casa. Você me diz que seu amigo mora na casa ao lado. Entenderei esse local porque tem relação com meu local atual. Mas se você me disser isso por telefone, não poderei encontrar a casa do seu amigo. Você precisará me fornecer o endereço completo.

Caminhos relativos especiais

O shell do Linux oferece maneiras de encurtar os caminhos durante a navegação. Para revelar os primeiros caminhos especiais, inserimos o comando `ls` com a flag `-a`. Essa flag modifica o comando `ls` para que *todos* os arquivos e diretórios sejam listados, incluindo os arquivos e diretórios ocultos:

```
user@hostname ~/Documents/Reports $ ls -a
.
..
```

```
report2018.txt
```

NOTE Consulte a página `man` de `ls` para entender o que `-a` está fazendo aqui.

Esse comando revelou dois resultados adicionais: Estes são caminhos especiais. Eles não representam novos arquivos ou diretórios, mas sim diretórios que você já conhece:

`.`
: Indica o *local atual* (neste caso, `Reports`).

`..`
: Indica o *diretório pai* (neste caso, `Documents`).

Geralmente, não é necessário usar o caminho relativo especial para o local atual. É mais fácil e compreensível digitar `report2018.txt` do que digitar `./Report2018.txt`. Mas o `.` tem usos que você aprenderá nas próximas seções. Por enquanto, vamos nos concentrar no caminho relativo para o diretório pai:

```
user@hostname ~/Documents/Reports $ cd ..
user@hostname ~/Documents $ pwd
/home/user/Documents
```

O exemplo de `cd` é muito mais fácil quando se usa `..` em vez do caminho absoluto. Além disso, podemos combinar esse padrão para subir rapidamente pela árvore de arquivos.

```
user@hostname ~/Documents $ cd ../..
$ pwd
/home
```

Exercícios Guiados

1. Identifique se cada um dos caminhos a seguir é *absoluto* ou *relativo*:

`/home/user/Downloads`	
`../Reports`	
`/var`	
`docs`	
`/`	

2. Observe a seguinte estrutura de arquivo. Nota: Os diretórios terminam com uma barra (`/`) quando `tree` é chamado com a opção `-F`. Você precisará de privilégios elevados para executar o comando `tree` no diretório root (`/`). O exemplo de saída a seguir é apenas demonstrativo e não é indicativo de uma estrutura de diretório completa. Use-o para responder às seguintes questões:

```
$ tree /

/
├── etc/
│   ├── network/
│   │   └── interfaces
│   ├── systemd/
│   │   ├── resolved.conf
│   │   ├── system/
│   │   ├── system.conf
│   │   ├── user/
│   │   └── user.conf
│   └── udev/
│       ├── rules.d/
│       └── udev.conf
└── home/
    ├── lost+found/
    └── user/
        └── Documents/

12 directories, 5 files
```

   Use essa estrutura para responder às questões a seguir.

Um usuário insere os seguintes comandos:

```
$ cd /etc/udev
$ ls -a
```

Qual será a saída do comando `ls -a`?

3. Digite o comando mais curto possível para cada um dos seguintes enunciados:
   - Seu local atual é root (`/`). Digite o comando para navegar até `lost+found` dentro do diretório `home` (exemplo):

     ```
     $ cd home/lost+found
     ```

   - Seu local atual é root (`/`). Digite o comando para navegar até o diretório `/etc/network/`.
   - Seu local atual é `/home/user/Documents/`. Navegue até o diretório `/etc/`.
   - Seu local atual é `/etc/systemd/system/`. Navegue até o diretório `/home/user/`.

4. Considere os seguintes comandos:

```
$ pwd
/etc/udev/rules.d
$ cd ../../systemd/user
$ cd ..
$ pwd
```

Qual a saída do comando `pwd` final?

Exercícios Exploratórios

1. Supondo que um usuário tenha inserido os seguintes comandos:

   ```
   $ mkdir "this is a test"
   $ ls
   this is a test
   ```

   Qual comando `cd` permitiria entrar nesse diretório?

2. Tente o mesmo exercício novamente, mas depois de digitar `cd this`, pressione a tecla TAB. O que é exibido no prompt?

   Isso é um exemplo de *autocompletar*, uma ferramenta inestimável não somente para poupar tempo como também para evitar erros de digitação.

3. Tente criar um diretório cujo nome contenha um caractere `\`. Exiba o nome do diretório com `ls` e remova o diretório.

Resumo

Nesta lição, você aprendeu:

- As bases do sistema de arquivos do Linux
- A diferença entre diretórios *pai* e *subdiretórios*
- A diferença entre caminhos de arquivo *absolutos* e *relativos*
- Os caminhos relativos especiais `.` e `..`
- Navegação no sistema de arquivos usando `cd`
- Mostrar seu local atual usando `pwd`
- Listar *todos* os arquivos e diretórios usando `ls -a`

Os seguintes comandos foram discutidos nesta lição:

`cd`
: Muda o diretório atual.

`pwd`
: Imprime o caminho do diretório de trabalho atual

`ls`
: Lista o conteúdo de um diretório e exibe as propriedades dos arquivos

`mkdir`
: Cria um novo diretório

`tree`
: Exibe uma lista hierárquica de uma árvore de diretórios

Respostas aos Exercícios Guiados

1. Identifique se cada um dos caminhos a seguir é *absoluto* ou *relativo*:

`/home/user/Downloads`	absoluto
`../Reports`	relativo
`/var`	absoluto
`docs`	relativo
`/`	absoluto

2. Observe a seguinte estrutura de arquivo. Nota: Os diretórios terminam com uma barra (`/`) quando `tree` é chamado com a opção `-F`. Você precisará de privilégios elevados para executar o comando `tree` no diretório root (`/`). O exemplo de saída a seguir é apenas demonstrativo e não é indicativo de uma estrutura de diretório completa. Use-o para responder às seguintes questões:

```
$ tree /

/
├── etc/
│   ├── network/
│   │   └── interfaces
│   ├── systemd/
│   │   ├── resolved.conf
│   │   ├── system/
│   │   ├── system.conf
│   │   ├── user/
│   │   └── user.conf
│   └── udev/
│       ├── rules.d/
│       └── udev.conf
└── home/
    ├── lost+found/
    └── user/
        └── Documents/

12 directories, 5 files
```

   Um usuário insere os seguintes comandos:

```
$ cd /etc/udev
$ ls -a
```

Qual será a saída do comando `ls -a`?

```
. .. rules.d udev.conf
```

3. Digite o comando mais curto possível para cada um dos seguintes enunciados:
   - Seu local atual é root (`/`). Digite o comando para navegar até `lost+found` dentro do diretório `home` (exemplo):

     ```
     $ cd home/lost+found
     ```

   - Seu local atual é root (`/`). Digite o comando para navegar até o diretório `/etc/network/`.

     ```
     $ cd etc/network
     ```

   - Seu local atual é `/home/user/Documents/`. Navegue até o diretório `/etc/`.

     ```
     $ cd /etc
     ```

   - Seu local atual é `/etc/systemd/system/`. Navegue até o diretório `/home/user/`.

     ```
     $ cd /home/user
     ```

4. Considere os seguintes comandos:

```
$ pwd
/etc/udev/rules.d
$ cd ../../systemd/user
$ cd ..
$ pwd
```

Qual a saída do comando `pwd` final?

```
/etc/systemd
```

Respostas aos Exercícios Exploratórios

1. Supondo que um usuário tenha inserido os seguintes comandos:

```
$ mkdir "this is a test"
$ ls
this is a test
```

   Qual comando `cd` permitiria entrar nesse diretório?

```
$ cd this\ is\ a\ test
```

2. Tente o mesmo exercício novamente, mas depois de digitar `cd this`, pressione a tecla TAB. O que é exibido no prompt?

```
$ cd this\ is\ a\ test
```

   Isso é um exemplo de *autocompletar*, uma ferramenta inestimável não somente para poupar tempo como também para evitar erros de digitação.

3. Tente criar um diretório cujo nome contenha um caractere `\`. Exiba o nome do diretório com `ls` e remova o diretório.

   É possível escapar da barra usando outra barra (`\\`) ou aspas simples ou duplas em torno do nome completo do diretório:

```
$ mkdir my\\dir
$ ls
'my\dir'
$ rmdir 'my\dir'
```

2.3 Lição 2

Certificação:	Linux Essentials
Versão:	1.6
Tópico:	2 Como se orientar em um sistema Linux
Objetivo:	2.3 Como usar diretórios e listar arquivos
Lição:	2 de 2

Introdução

O sistema operacional Unix foi originalmente projetado para computadores mainframe em meados da década de 1960. Esses computadores eram compartilhados entre muitos usuários, que acessavam os recursos do sistema através de *terminais*. Essas idéias fundamentais ainda estão presentes nos sistemas Linux de hoje em dia. Ainda falamos sobre o uso de “terminais” para inserir comandos no shell, e os sistemas Linux são organizados de tal maneira que é fácil criar diversos usuários em um único sistema.

Diretórios Home

Eis um exemplo de um sistema de arquivos normal no Linux:

```
$ tree -L 1 /
/
├── bin
├── boot
├── cdrom
├── dev
```

```
├── etc
├── home
├── lib
├── mnt
├── opt
├── proc
├── root
├── run
├── sbin
├── srv
├── sys
├── tmp
├── usr
└── var
```

A maioria desses diretórios existe em todos os sistemas Linux. De servidores a supercomputadores, passando por pequenos sistemas embarcados, um usuário experiente do Linux pode ter a certeza de que vai encontrar o comando `ls` dentro de `/bin`, que pode alterar a configuração do sistema modificando arquivos em `/etc` e ler os logs do sistema em `/var`. A localização padrão desses arquivos e diretórios é definida pelo FHS (Filesystem Hierarchy Standard, ou padrão de sistema de arquivos hierárquico), que será discutido em uma lição mais pra frente. Você aprenderá mais sobre o conteúdo desses diretórios à medida que continuar estudando o Linux, mas por enquanto, é importante saber que:

- as alterações feitas no sistema de arquivos raiz (root) afetarão todos os usuários, e
- alterar arquivos no sistema de arquivos raiz exigirá permissões de administrador.

Isso significa que os usuários normais não terão a permissão para alterar esses arquivos, e talvez nem mesmo para a leitura deles. Abordaremos o tópico das permissões em uma seção posterior.

Agora, vamos nos concentrar no diretório `/home`, que você já deve conhecer bem:

```
$ tree -L 1 /home
/home
├── user
├── michael
└── lara
```

Nosso sistema de exemplo tem três usuários normais e cada um deles tem acesso a seu próprio local dedicado, no qual é possível criar e modificar arquivos e diretórios sem afetar o vizinho. Por exemplo, na lição anterior, estávamos trabalhando com a seguinte estrutura de arquivos:

```
$ tree /home/user
user
└── Documents
    ├── Mission-Statement
    └── Reports
        └── report2018.txt
```

Na realidade, o sistema de arquivos pode ser assim:

```
$ tree /home
/home
├── user
│   └── Documents
│       ├── Mission-Statement
│       └── Reports
│           └── report2018.txt
├── michael
│   ├── Documents
│   │   └── presentation-for-clients.odp
│   └── Music
```

...e assim por diante para `lara`.

No Linux, `/home` é semelhante a um prédio de apartamentos. Muitos usuários podem ter seu espaço aqui, separados em apartamentos dedicados. As instalações básicas e a manutenção do próprio edifício são de responsabilidade do usuário root, que representa o síndico.

O caminho relativo especial para home

Ao iniciar uma nova sessão no terminal do Linux, vemos um prompt de comando semelhante a este:

```
user@hostname ~ $
```

O til (`~`) representa, aqui, o diretório *home*. Se executarmos o comando `ls`, veremos um resultado conhecido:

```
$ cd ~
$ ls
```

```
Documents
```

Compare-o ao sistema de arquivos acima para entender melhor.

Considere agora o que sabemos sobre o Linux: ele é semelhante a um prédio de apartamentos, com muitos usuários residentes em `/home`. Portanto, a casa de `user` será diferente da casa do usuário `michael`. Para demonstrar isso, usaremos o comando `su` para *trocar de usuário*.

```
user@hostname ~ $ pwd
/home/user
user@hostname ~ $ su - michael
Password:
michael@hostname ~ $ pwd
/home/michael
```

O significado de `~` muda dependendo de quem for o usuário. Para `michael`, o caminho absoluto de `~` é `/home/michael`. Para `lara`, o caminho absoluto de `~` é `/home/lara`, e assim por diante.

Caminhos de arquivos relativos-a-home

A possibilidade de usar `~` para os comandos é muito conveniente, desde que você não troque de usuário. Vamos considerar o exemplo a seguir para `user`, que iniciou uma nova sessão:

```
$ ls
Documents
$ cd Documents
$ ls
Mission-Statement
Reports
$ cd Reports
$ ls
report2018.txt
$ cd ~
$ ls
Documents
```

Note que os usuários sempre iniciam uma nova sessão em seu diretório pessoal. Neste exemplo, `user` trocou para o subdiretório `Documents/Reports` e, com o comando `cd ~`, voltou ao ponto de partida. É possível executar a mesma ação usando o comando `cd` sem argumentos:

```
$ cd Documents/Reports
$ pwd
/home/user/Documents/Reports
$ cd
$ pwd
/home/user
```

E por fim: podemos especificar os diretórios pessoais de *outros usuários* especificando o nome de usuário após o til. Por exemplo:

```
$ ls ~michael
Documents
Music
```

Porém, isso só funciona se `michael` nos der permissão para visualizar o conteúdo de seu diretório pessoal.

Vamos considerar uma situação em que `michael` gostaria de visualizar o arquivo `report2018.txt` no diretório inicial de `user`. Supondo que `michael` tenha permissão para fazê-lo, ele pode usar o comando `less`.

```
$ less ~user/Documents/Reports/report2018.txt
```

Qualquer caminho de arquivo que contenha o caractere `~` é chamado de caminho *relativo-a-home*.

Arquivos e diretórios ocultos

Na lição anterior, introduzimos a opção `-a` para o comando `ls`. Usamos `ls -a` para introduzir os dois caminhos relativos especiais: `.` e `..`. A opção `-a` lista todos os arquivos e diretórios, incluindo os arquivos e diretórios *ocultos*.

```
$ ls -a ~
.
..
.bash_history
.bash_logout
.bash-profile
.bashrc
Documents
```

Os arquivos e diretórios ocultos sempre começam com um ponto (`.`). Por padrão, o diretório home de um usuário inclui diversos arquivos ocultos. Eles servem geralmente para estabelecer as configurações específicas do usuário e devem ser modificados somente por um usuário experiente.

A opção de lista longa

Existem muitas opções que permitem mudar o comportamento do comando `ls`. Vamos conhecer uma das mais comuns:

```
$ ls -l
-rw-r--r-- 1 user staff      3606 Jan 13  2017 report2018.txt
```

`-l` cria uma *lista longa*. Os arquivos e diretórios ocuparão uma linha cada, mas também serão exibidas informações adicionais sobre cada arquivo e diretório.

`-rw-r--r--`

Tipo de arquivo e permissões do arquivo. Note que um arquivo regular começa com traço e um diretório começa com `d`.

`1`

Número de links para o arquivo.

`user staff`

Especifica a posse do arquivo. `user` é o proprietário do arquivo, o qual também está associado ao grupo `staff`.

`3606`

Tamanho do arquivo em bytes.

`Jan 13 2017`

Registro de data e hora da última modificação do arquivo.

`report2018.txt`

Nome do arquivo.

Temas como posse, permissões e links serão abordados em lições futuras. Como vemos, a versão em lista longa do `ls` é muitas vezes preferível à padrão.

Opções adicionais de ls

Eis algumas das maneiras mais comuns de se usar o comando `ls`. O usuário pode combinar várias opções para obter a saída desejada.

`ls -lh`

A combinação de *lista longa* com tamanhos de arquivo *legíveis para humanos* fornece sufixos úteis como `M` para megabytes ou `K` para quilobytes.

`ls -d */`

A opção `-d` lista os diretórios, mas não seu conteúdo. A combinação com `*/` mostra apenas os subdiretórios e nenhum arquivo.

`ls -lt`

Combina a *lista longa* com a opção de classificar por *data de modificação*. Os arquivos com as alterações mais recentes aparecem na parte superior e os arquivos com as alterações mais antigas, na parte inferior. Mas essa ordem pode ser invertida com:

`ls -lrt`

Combina a *lista longa* com *classificar por data (de modificação)*, junto com `-r`, que *inverte* a classificação. Agora, os arquivos com as alterações mais recentes aparecem na parte inferior da lista. Além de classificar por data de *modificação*, os arquivos também podem ser classificados por data de *acesso* ou data de alteração de *status*.

`ls -lX`

Combina a *lista longa* com a opção de classificar por *eXtensão de arquivo*. Serve, por exemplo, para agrupar todos os arquivos que terminam com `.txt`, todos os arquivos que terminam com `.jpg` e assim por diante.

`ls -S`

O `-S` classifica por *tamanho* de arquivo, da mesma forma que `-t` e `-X` classificam por data e extensão, respectivamente. Os arquivos maiores aparecerão primeiro e os menores, por último. Observe que o conteúdo dos subdiretórios *não* está incluído na classificação.

`ls -R`

A opção `-R` modifica o comando `ls` para exibir uma lista *recursiva*. O que isso significa?

A recursão no Bash

A recursão (ou recursividade) é uma situação em que “algo é definido em termos de si mesmo”. A

recursão é um conceito importantíssimo nas ciências da computação, mas aqui seu significado é bem mais simples. Vamos considerar nosso exemplo de antes:

```
$ ls ~
Documents
```

Já sabemos que `user` possui um diretório inicial (home) e que nesse diretório existe um subdiretório. Até agora, o `ls` nos mostrou apenas os arquivos e subdiretórios de um local, sem informar qual o conteúdo desses subdiretórios. Nestas lições, usamos o comando `tree` para exibir o conteúdo de vários diretórios. Infelizmente, o `tree` não é um dos utilitários integrantes do Linux e, portanto, nem sempre está disponível. Compare a saída de `tree` com a saída de `ls -R` nos seguintes exemplos:

```
$ tree /home/user
user
└── Documents
    ├── Mission-Statement
    └── Reports
        └── report2018.txt

$ ls -R ~
/home/user/:
Documents

/home/user/Documents:
Mission-Statement
Reports

/home/user/Documents/Reports:
report2018.txt
```

Como você pode ver, com a opção recursiva, temos uma lista muito maior de arquivos. Na verdade, é como se rodássemos o comando `ls` no diretório inicial de `user` e encontrássemos um subdiretório. Entramos então nesse subdiretório e executamos o comando `ls` novamente. Encontramos o arquivo `Mission-Statement` e outro subdiretório chamado `Reports`. E, novamente, entramos nesse subdiretório e executamos o comando `ls` outra vez. Basicamente, executar o comando `ls -R` é como dizer ao Bash: “Execute o `ls` aqui e repita o comando em todos os subdiretórios que encontrar”.

A recursividade é particularmente importante nos comandos de modificação de arquivos, como copiar ou remover diretórios. Por exemplo, se você quiser copiar o subdiretório `Documents`,

precisará especificar uma cópia recursiva para estender esse comando a todos os subdiretórios.

Exercícios Guiados

1. Use a estrutura de arquivos abaixo para responder às três questões a seguir:

```
/
├── etc/
│   ├── network/
│   │   └── interfaces/
│   ├── systemd/
│   │   ├── resolved.conf
│   │   ├── system/
│   │   ├── system.conf
│   │   ├── user/
│   │   └── user.conf
│   └── udev/
│       ├── rules.d
│       └── udev.conf
└── home/
    ├── lost+found/
    ├── user/
    │   └── Documents/
    └── michael/
        └── Music/
```

   - Qual comando permite navegar até o diretório `network` independentemente de seu local atual?

   - Qual comando `user` poderia digitar para navegar até seu diretório `Documents` a partir de `/etc/udev`? Use o caminho mais curto possível.

   - Qual comando `user` poderia digitar para navegar até o diretório `Music` de `michael`? Use o caminho mais curto possível.

2. Considere a seguinte saída de `ls -lh` para responder às duas questões a seguir. Note que os diretórios são indicados com um `d` no início da linha.

```
drwxrwxrwx  5 eric eric  4.0K Apr 26  2011 China/
-rwxrwxrwx  1 eric eric  1.5M Jul 18  2011 img_0066.jpg
```

```
-rwxrwxrwx  1 eric eric  1.5M Jul 18  2011 img_0067.jpg
-rwxrwxrwx  1 eric eric  1.6M Jul 18  2011 img_0074.jpg
-rwxrwxrwx  1 eric eric  1.8M Jul 18  2011 img_0075.jpg
-rwxrwxrwx  1 eric eric   46K Jul 18  2011 scary.jpg
-rwxrwxrwx  1 eric eric  469K Jan 29  2018 Screenshot from 2017-08-13 21-22-24.png
-rwxrwxrwx  1 eric eric  498K Jan 29  2018 Screenshot from 2017-08-14 21-18-07.png
-rwxrwxrwx  1 eric eric  211K Jan 29  2018 Screenshot from 2018-01-06 23-29-30.png
-rwxrwxrwx  1 eric eric  150K Jul 18  2011 tobermory.jpg
drwxrwxrwx  6 eric eric  4.0K Apr 26  2011 Tokyo/
-rwxrwxrwx  1 eric eric  1.4M Jul 18  2011 Toronto 081.jpg
-rwxrwxrwx  1 eric eric  1.4M Jul 18  2011 Toronto 085.jpg
-rwxrwxrwx  1 eric eric  944K Jul 18  2011 Toronto 152.jpg
-rwxrwxrwx  1 eric eric  728K Jul 18  2011 Toronto 173.jpg
drwxrwxrwx  2 eric eric  4.0K Jun  5  2016 Wallpapers/
```

- Ao rodar o comando `ls -lrS`, qual arquivo aparecerá no início?

- Descreva o que você espera ver como saída de `ls -ad */`.

Exercícios Exploratórios

1. Rode o comando `ls -lh` num diretório que contenha subdiretórios. Observe o tamanho listado desses diretórios. Esses tamanhos de arquivo parecem corretos? Eles representam com precisão o conteúdo de todos os arquivos contidos naquele diretório?

2. Eis um novo comando a experimentar: `du -h`. Rode esse comando e descreva a saída resultante.

3. Em muitos sistemas Linux, podemos digitar `ll` e obter a mesma saída que teríamos com `ls -l`. Porém, note que `ll` não é um comando. Por exemplo, `man ll` exibirá a mensagem de que não existe uma página de manual para esse caso. Este é um exemplo de *alias*. Por que os aliases seriam úteis para um usuário?

Resumo

Nesta lição, você aprendeu:

- que cada usuário do Linux tem um diretório home,
- o diretório home do usuário atual pode ser acessado usando `~`,
- qualquer caminho de arquivo que use `~` é chamado de caminho *relativo-a-home*.

Você também aprendeu algumas das maneiras mais comuns de modificar o comando `ls`.

`-a` (all)
imprime todos os arquivos/diretórios, incluindo os ocultos

`-d` (directories)
lista os diretórios, mas não seu conteúdo

`-h` (human readable)
imprime o tamanho dos arquivos em formato legível para humanos

`-l` (long list)
fornece detalhes extras, um arquivo/diretório por linha

`-r` (reverse)
inverte a ordem de uma classificação

`-R` (recursive)
lista todos os arquivos, incluindo os que estão em cada subdiretório

`-S` (size)
classifica por tamanho de arquivo

`-t` (time)
classifica por data de modificação

`-X` (eXtension)
classifica por extensão de arquivo

Respostas aos Exercícios Guiados

1. Use a estrutura de arquivos abaixo para responder às três questões a seguir:

```
/
├── etc/
│   ├── network/
│   │   └── interfaces/
│   ├── systemd/
│   │   ├── resolved.conf
│   │   ├── system/
│   │   ├── system.conf
│   │   ├── user/
│   │   └── user.conf
│   └── udev/
│       ├── rules.d
│       └── udev.conf
└── home/
    ├── lost+found/
    ├── user/
    │   └── Documents/
    └── michael/
        └── Music/
```

   - Qual comando permite navegar até o diretório `network` independentemente de seu local atual?

     ```
     cd /etc/network
     ```

   - Qual comando `user` poderia digitar para navegar até seu diretório `Documents` a partir de `/etc/udev`? Use o caminho mais curto possível.

     ```
     cd ~/Documents
     ```

   - Qual comando `user` poderia digitar para navegar até o diretório `Music` de `michael`? Use o caminho mais curto possível.

     ```
     cd ~michael/Music
     ```

2. Considere a seguinte saída de `ls -lh` para responder às duas questões a seguir. Note que os diretórios são indicados com um `d` no início da linha.

```
drwxrwxrwx  5 eric eric  4.0K Apr 26  2011 China/
-rwxrwxrwx  1 eric eric  1.5M Jul 18  2011 img_0066.jpg
-rwxrwxrwx  1 eric eric  1.5M Jul 18  2011 img_0067.jpg
-rwxrwxrwx  1 eric eric  1.6M Jul 18  2011 img_0074.jpg
-rwxrwxrwx  1 eric eric  1.8M Jul 18  2011 img_0075.jpg
-rwxrwxrwx  1 eric eric   46K Jul 18  2011 scary.jpg
-rwxrwxrwx  1 eric eric  469K Jan 29  2018 Screenshot from 2017-08-13 21-22-24.png
-rwxrwxrwx  1 eric eric  498K Jan 29  2018 Screenshot from 2017-08-14 21-18-07.png
-rwxrwxrwx  1 eric eric  211K Jan 29  2018 Screenshot from 2018-01-06 23-29-30.png
-rwxrwxrwx  1 eric eric  150K Jul 18  2011 tobermory.jpg
drwxrwxrwx  6 eric eric  4.0K Apr 26  2011 Tokyo/
-rwxrwxrwx  1 eric eric  1.4M Jul 18  2011 Toronto 081.jpg
-rwxrwxrwx  1 eric eric  1.4M Jul 18  2011 Toronto 085.jpg
-rwxrwxrwx  1 eric eric  944K Jul 18  2011 Toronto 152.jpg
-rwxrwxrwx  1 eric eric  728K Jul 18  2011 Toronto 173.jpg
drwxrwxrwx  2 eric eric  4.0K Jun  5  2016 Wallpapers/
```

   - Ao rodar o comando `ls -lrS`, qual arquivo aparecerá no início?

     Todas as pastas têm 4.0K, que é o menor tamanho de arquivo. Nesse caso, `ls` classifica alfabeticamente por padrão. A resposta correta é o diretório `scary.jpg`

   - Descreva o que você espera ver como saída de `ls -ad */`.

     Esse comando exibe todos os subdiretórios, incluindo subdiretórios ocultos.

Respostas aos Exercícios Exploratórios

1. Rode o comando `ls -lh` num diretório que contenha subdiretórios. Observe o tamanho listado desses diretórios. Esses tamanhos de arquivo parecem corretos? Eles representam com precisão o conteúdo de todos os arquivos contidos naquele diretório?

   A resposta é não. Cada diretório tem um tamanho de arquivo listado como 4096 bytes. Isso acontece porque os diretórios, aqui, são uma abstração: eles não existem em forma de estrutura em árvore no disco. Quando vemos um diretório listado, na verdade estamos vendo um *link* para uma lista de arquivos. O tamanho desses links é de 4096 bytes.

2. Eis um novo comando a experimentar: `du -h`. Rode esse comando e descreva a saída resultante.

   O comando `du` gera uma lista com todos os arquivos e diretórios e indica o tamanho de cada um. Por exemplo, `du -s` exibe o tamanho de todos os arquivos, diretórios e subdiretórios em um determinado local.

3. . Em muitos sistemas Linux, podemos digitar `ll` e obter a mesma saída que teríamos com `ls -l`. Porém, note que `ll` não é um comando. Por exemplo, `man ll` exibirá a mensagem de que não existe uma página de manual para esse caso. Você acha que isso se relaciona a qual recurso da linha de comando?

   `ll` é um *alias* de `ls -l`. No Bash, podemos usar aliases para simplificar comandos usados com frequência. `ll` costuma já vir definido no Linux, mas também é possível criar os seus.

2.4 Criando, Movendo e Deletando Arquivos

Referência ao LPI objectivo

Linux Essentials version 1.6, Exam 010, Objective 2.4

Peso

2

Áreas chave de conhecimento

- Arquivos e diretórios
- Uso de maiúsculas e minúsculas
- Englobamento e uso de aspas

Segue uma lista parcial de arquivos, termos e utilitários utilizados

- `mv`, `cp`, `rm`, `touch`
- `mkdir`, `rmdir`

2.4 Lição 1

Certificação:	Linux Essentials
Versão:	1.6
Tópico:	2 Como se orientar em um sistema Linux
Objetivo:	2.4 Criar, mover e remover arquivos
Lição:	1 de 1

Introdução

Esta lição aborda o gerenciamento de arquivos e diretórios no Linux usando ferramentas de linha de comando.

Um arquivo é uma coleção de dados com um nome e um conjunto de atributos. Se, por exemplo, você transferisse algumas fotos de seu telefone para um computador e lhes desse nomes descritivos, você teria uma porção de arquivos de imagem no computador. Esses arquivos têm atributos, como a data e hora em que o arquivo foi acessado ou modificado pela última vez.

Um diretório é um tipo especial de arquivo usado para organizar arquivos. Uma boa maneira de pensar em diretórios é compará-los às pastas suspensas usadas para organizar papéis em um gaveteiro metálico. Porém, ao contrário das pastas de documentos em papel, podemos facilmente colocar diretórios dentro de outros diretórios.

A linha de comando é a maneira mais eficaz de gerenciar arquivos em um sistema Linux. As ferramentas da linha de comando e do shell têm recursos que tornam o uso da linha de comando mais rápido e fácil do que num gerenciador de arquivos gráfico.

Nesta seção, usaremos os comandos `ls`, `mv`, `cp`, `pwd`, `find`, `touch`, `rm`, `rmdir`, `echo`, `cat` e `mkdir`

para gerenciar e organizar arquivos e diretórios.

Maiúsculas e minúsculas

Ao contrário do Microsoft Windows, os nomes de arquivos e diretórios nos sistemas Linux diferenciam maiúsculas de minúsculas. Isso significa que os nomes `/etc/` e `/ETC/` designam diretórios diferentes. Experimente os seguintes comandos:

```
$ cd /
$ ls
bin   dev  home  lib64  mnt  proc  run   srv  tmp  var
boot  etc  lib   media  opt  root  sbin  sys  usr
$ cd ETC
bash: cd: ETC: No such file or directory
$ pwd
/
$ cd etc
$ pwd
/etc
```

O `pwd` mostra o diretório em que você está no momento. Como vemos, tentar mudar para `/ETC` não funcionou, pois esse diretório não existe. Mas quando mudamos para o diretório `/etc`, que existe, a operação foi bem-sucedida.

Criando diretórios

O comando `mkdir` é usado para criar diretórios.

Vamos criar um novo diretório dentro do nosso diretório pessoal:

```
$ cd ~
$ pwd
/home/user
$ ls
Desktop  Documents  Downloads
$ mkdir linux_essentials-2.4
$ ls
Desktop  Documents  Downloads  linux_essentials-2.4
$ cd linux_essentials-2.4
$ pwd
/home/emma/linux_essentials-2.4
```

Ao longo de toda esta lição, todos os comandos serão dados neste diretório ou em um de seus subdiretórios.

Para retornar facilmente ao diretório da lição a partir de qualquer outra posição em seu sistema de arquivos, use o comando:

```
$ cd ~/linux_essentials-2.4
```

O shell interpreta o caracter `~` como seu diretório inicial (home).

Quando estiver no diretório da lição, crie mais alguns diretórios para usar nos exercícios. Podemos adicionar todos os nomes de diretório, separados por espaços, ao `mkdir`:

```
$ mkdir creating moving copying/files copying/directories deleting/directories
deleting/files globs
mkdir: cannot create directory ‘copying/files’: No such file or directory
mkdir: cannot create directory ‘copying/directories’: No such file or directory
mkdir: cannot create directory ‘deleting/directories’: No such file or directory
mkdir: cannot create directory ‘deleting/files’: No such file or directory
$ ls
creating  globs  moving
```

Observe a mensagem de erro e que apenas `moving`, `globs` e `creating` foram criados. Os diretórios `copying` e `deleting` ainda não existem. O `mkdir`, por padrão, não cria um diretório dentro de um diretório que não existe. A opção `-p` ou `--parents` instrui o `mkdir` a criar diretórios pai caso eles não existam. Experimente o mesmo comando `mkdir` com a opção `-p`:

```
$ mkdir -p creating moving copying/files copying/directories deleting/directories
deleting/files globs
```

Desta vez, não apareceu nenhuma mensagem de erro. Vamos ver quais diretórios existem agora:

```
$ find
.
./creating
./moving
./globs
./copying
./copying/files
./copying/directories
```

```
./deleting
./deleting/directories
./deleting/files
```

O programa `find` é geralmente usado para procurar arquivos e diretórios, mas quando não definimos nenhuma opção, ele mostra uma lista de todos os arquivos, diretórios e subdiretórios do seu diretório atual.

TIP Ao listar o conteúdo de um diretório com `ls`, as opções `-t` e `-r` são particularmente úteis. Elas ordenam a saída por tempo (`-t`) e invertem a ordem de classificação (`-r`). Nesse caso, os arquivos mais recentes estarão na parte inferior da saída.

Criando arquivos

Normalmente, os arquivos são criados pelos programas que trabalham com os dados armazenados nos arquivos. Um arquivo vazio pode ser criado usando o comando `touch`. Se executarmos o `touch` em um arquivo existente, o conteúdo do arquivo não será alterado, mas o registro de data e hora de modificação do arquivo será atualizado.

Execute o seguinte comando para criar alguns arquivos para a lição de globbing:

```
$ touch globs/question1 globs/question2012 globs/question23 globs/question13
globs/question14
$ touch globs/star10 globs/star1100 globs/star2002 globs/star2013
```

Em seguida, vamos verificar se todos os arquivos existem no diretório `globs`:

```
$ cd globs
$ ls
question1   question14    question23  star1100  star2013
question13  question2012  star10      star2002
```

Viu como o `touch` criou os arquivos? Podemos visualizar o conteúdo de um arquivo de texto com o comando `cat`. Experimente em um dos arquivos que acabamos de criar:

```
$ cat question14
```

Como `touch` cria arquivos vazios, não devem aparecer resultados. Podemos usar `echo` com `>` para criar arquivos de texto simples. Experimente:

```
$ echo hello > question15
$ cat question15
hello
```

`echo` exibe texto na linha de comando. O caractere `>` instrui o shell a escrever a saída de um comando no arquivo especificado, em vez do terminal. Isso faz com que a saída de `echo`, neste caso `hello`, seja gravada no arquivo `question15`. Isso não é específico ao `echo`, podendo ser usado com qualquer comando.

WARNING Cuidado ao usar `>`! Se já existir um arquivo com aquele nome, ele será sobrescrito!

Renomeando arquivos

O comando `mv` serve para mover e renomear arquivos.

Defina `moving` como seu diretório de trabalho:

```
$ cd ~/linux_essentials-2.4/moving
```

Crie alguns arquivos para praticar. Neste momento, você já deve estar familiarizado com estes comandos:

```
$ touch file1 file22
$ echo file3 > file3
$ echo file4 > file4
$ ls
file1  file22  file3  file4
```

Suponha que `file22` seja um erro de digitação; o correto seria `file2`. Corrija o nome com o comando `mv`. Ao renomear um arquivo, o primeiro argumento é o nome atual, o segundo é o novo nome:

```
$ mv file22 file2
$ ls
file1  file2  file3  file4
```

Tenha cuidado ao usar o comando `mv`. Se o novo nome já estiver atribuído a um arquivo existente, ele será sobrescrito. Vamos testar com `file3` e `file4`:

```
$ cat file3
file3
$ cat file4
file4
$ mv file4 file3
$ cat file3
file4
$ ls
file1  file2  file3
```

Note que o conteúdo de `file3` agora está em `file4`. Use a opção `-i` para que `mv` avise caso você esteja prestes a sobrescrever um arquivo existente. Experimente:

```
$ touch file4 file5
$ mv -i file4 file3
mv: overwrite ‘file3’? y
```

Movendo arquivos

Os arquivos são movidos de um diretório para outro com o comando `mv`.

Crie alguns diretórios para os quais moveremos arquivos:

```
$ cd ~/linux_essentials-2.4/moving
$ mkdir dir1 dir2
$ ls
dir1  dir2  file1  file2  file3  file5
```

Mova `file1` para `dir1`:

```
$ mv file1 dir1
$ ls
dir1  dir2  file2  file3  file5
$ ls dir1
file1
```

Observe como o último argumento para `mv` é o diretório de destino. Sempre que o último argumento para `mv` for um diretório, os arquivos serão movidos para ele. É possivel especificar diversos arquivos em um único comando `mv`:

```
$ mv file2 file3 dir2
$ ls
dir1  dir2  file5
$ ls dir2
file2  file3
```

Também podemos usar o `mv` para mover e renomear diretórios. Renomeie `dir1` como `dir3`:

```
$ ls
dir1  dir2  file5
$ ls dir1
file1
$ mv dir1 dir3
$ ls
dir2  dir3  file5
$ ls dir3
file1
```

Excluindo arquivos e diretórios

O comando `rm` serve para excluir arquivos e diretórios, ao passo que o comando `rmdir` pode excluir apenas diretórios. Vamos limpar o diretório `moving` excluindo `file5`:

```
$ cd ~/linux_essentials-2.4/moving
$ ls
dir2  dir3  file5
$ rmdir file5
rmdir: failed to remove ‘file5’: Not a directory
$ rm file5
$ ls
dir2  dir3
```

Por padrão, o `rmdir` pode excluir apenas diretórios vazios; portanto, tivemos de usar `rm` para excluir um arquivo comum. Tente excluir o diretório `deleting`:

```
$ cd ~/linux_essentials-2.4/
$ ls
copying  creating  deleting  globs  moving
$ rmdir deleting
rmdir: failed to remove ‘deleting’: Directory not empty
```

```
$ ls -l deleting
total 0
drwxrwxr-x. 2 emma emma 6 Mar 26 14:58 directories
drwxrwxr-x. 2 emma emma 6 Mar 26 14:58 files
```

Como vimos, o `rmdir` se recusa a excluir um diretório que não esteja vazio. Use `rmdir` para remover um dos subdiretórios vazios do diretório `deleting`:

```
$ ls -a deleting/files
.  ..
$ rmdir deleting/files
$ ls -l deleting
directories
```

Excluir um grande número de arquivos ou estruturas de diretórios profundas, com muitos subdiretórios, pode parecer uma tarefa entediante, mas na verdade é fácil. Por padrão, o `rm` funciona apenas em arquivos regulares. A opção `-r` é usada para anular esse comportamento. Tenha cuidado, `rm -r` pode acabar sendo um tiro no pé! Quando usamos a opção `-r`, o `rm` não somente exclui qualquer diretório, como também tudo o que estiver dentro desse diretório, incluindo os subdiretórios e seu conteúdo. Veja você mesmo como o `rm -r` funciona:

```
$ ls
copying  creating  deleting  globs  moving
$ rm deleting
rm: cannot remove ‘deleting’: Is a directory
$ ls -l deleting
total 0
drwxrwxr-x. 2 emma emma 6 Mar 26 14:58 directories
$ rm -r deleting
$ ls
copying  creating  globs  moving
```

Percebeu que `deleting` se foi, mesmo não estando vazio? Como o `mv`, o `rm` tem uma opção `-i` para exibir um aviso antes de executar qualquer ação. Use `rm -ri` para remover os diretórios da seção `moving` que não forem mais necessários:

```
$ find
.
./creating
./moving
```

```
./moving/dir2
./moving/dir2/file2
./moving/dir2/file3
./moving/dir3
./moving/dir3/file1
./globs
./globs/question1
./globs/question2012
./globs/question23
./globs/question13
./globs/question14
./globs/star10
./globs/star1100
./globs/star2002
./globs/star2013
./globs/question15
./copying
./copying/files
./copying/directories
$ rm -ri moving
rm: descend into directory ‘moving’? y
rm: descend into directory ‘moving/dir2’? y
rm: remove regular empty file ‘moving/dir2/file2’? y
rm: remove regular empty file ‘moving/dir2/file3’? y
rm: remove directory ‘moving/dir2’? y
rm: descend into directory ‘moving/dir3’? y
rm: remove regular empty file ‘moving/dir3/file1’? y
rm: remove directory ‘moving/dir3’? y
rm: remove directory ‘moving’? y
```

Copiando arquivos e diretórios

O comando `cp` é usado para copiar arquivos e diretórios. Copie alguns arquivos para o diretório `copying`:

```
$ cd ~/linux_essentials-2.4/copying
$ ls
directories  files
$ cp /etc/nsswitch.conf files/nsswitch.conf
$ cp /etc/issue /etc/hostname files
```

Se o último argumento for um diretório, o `cp` criará uma cópia dos argumentos anteriores dentro

do diretório. Como no caso de `mv`, vários arquivos podem ser especificados ao mesmo tempo, desde que o destino seja um diretório.

Quando os dois operandos de `cp` são arquivos e os dois arquivos existem, o `cp` sobrescreve o segundo arquivo com uma cópia do primeiro arquivo. Para comprovar, vamos substituir o arquivo `issue` pelo arquivo `hostname`:

```
$ cd ~/linux_essentials-2.4/copying/files
$ ls
hostname  issue  nsswitch.conf
$ cat hostname
mycomputer
$ cat issue
Debian GNU/Linux 9 \n \l

$ cp hostname issue
$ cat issue
mycomputer
```

Agora, vamos tentar criar uma cópia do diretório `files` dentro do diretório `directory`:

```
$ cd ~/linux_essentials-2.4/copying
$ cp files directories
cp: omitting directory ‘files’
```

Como vemos, o `cp`, por padrão, só funciona em arquivos individuais. Para copiar um diretório, usamos a opção `-r`. Lembre-se de que a opção `-r` fará com que `cp` também copie o conteúdo do diretório que está sendo copiado:

```
$ cp -r files directories
$ find
.
./files
./files/nsswitch.conf
./files/fstab
./files/hostname
./directories
./directories/files
./directories/files/nsswitch.conf
./directories/files/fstab
./directories/files/hostname
```

Percebeu que, quando um diretório existente é usado como destino, o `cp` cria uma cópia do diretório de origem dentro dele? Se o destino não existir, ele será criado e preenchido com o conteúdo do diretório de origem:

```
$ cp -r files files2
$ find
.
./files
./files/nsswitch.conf
./files/fstab
./files/hostname
./directories
./directories/files
./directories/files/nsswitch.conf
./directories/files/fstab
./directories/files/hostname
./files2
./files2/nsswitch.conf
./files2/fstab
./files2/hostname
```

Globbing

Costumamos chamar de globbing uma linguagem simples de correspondência de padrões. Os shells da linha de comando nos sistemas Linux usam essa linguagem para buscar grupos de arquivos cujos nomes correspondam a um padrão específico. O POSIX.1-2017 especifica os seguintes padrões para a correspondência de caracteres:

`*`
: Corresponde a qualquer número de quaisquer caracteres, incluindo nenhum caractere

`?`
: Corresponde a qualquer caractere

`[]`
: Corresponde a uma classe de caracteres

Em bom português, isso significa que você pode dizer ao seu shell para procurar por um padrão, em vez de uma sequência literal de texto. Geralmente, os usuários do Linux especificam diversos arquivos com um glob em vez de digitar o nome de cada arquivo. Execute os seguintes comandos:

```
$ cd ~/linux_essentials-2.4/globs
$ ls
question1   question14  question2012  star10    star2002
question13  question15  question23    star1100  star2013
$ ls star1*
star10  star1100
$ ls star*
star10  star1100  star2002  star2013
$ ls star2*
star2002  star2013
$ ls star2*2
star2002
$ ls star2013*
star2013
```

O shell expande `*` para qualquer número de qualquer coisa, e assim ele interpreta `star*` como sendo qualquer coisa, dentro do contexto relevante, que comece com `star`. Quando você executa o comando `ls star*`, o shell não executa o programa `ls` com um argumento `star*`, mas sim procura por arquivos no diretório atual que correspondam ao padrão `star*`(incluindo apenas `star`), e transforma cada arquivo correspondente ao padrão em um argumento para `ls`:

```
$ ls star*
```

No que diz respeito ao `ls`, isso é o equivalente a

```
$ ls star10  star1100  star2002  star2013
```

O caractere `*` não significa nada para `ls`. Para comprovar, execute o seguinte comando:

```
$ ls star\*
ls: cannot access star*: No such file or directory
```

Quando colocamos `\` antes de um caractere, estamos instruindo seu shell a não interpretá-lo. Neste caso, queremos que `ls` tenha um argumento `star*`, em vez da expansão feita pelo glob `star*`.

O `?` expande para qualquer caractere único. Experimente os seguintes comandos para ver por si mesmo:

```
$ ls
question1   question14  question2012  star10    star2002
question13  question15  question23    star1100  star2013
$ ls question?
question1
$ ls question1?
question13  question14  question15
$ ls question?3
question13  question23
$ ls question13?
ls: cannot access question13?: No such file or directory
```

Os colchetes `[]` são usados para combinar faixas ou classes de caracteres. Os colchetes `[]` funcionam como nas expressões regulares POSIX, exceto porque, nos globs, o `^` é usado no lugar de `!`.

Crie alguns arquivos para experimentar:

```
$ mkdir brackets
$ cd brackets
$ touch file1 file2 file3 file4 filea fileb filec file5 file6 file7
```

Os intervalos entre colchetes `[]` são expressos usando um `-`:

```
$ ls
file1  file2  file3  file4  file5  file6  file7  filea  fileb  filec
$ ls file[1-2]
file1  file2
$ ls file[1-3]
file1  file2  file3
```

É possível especificar vários intervalos:

```
$ ls file[1-25-7]
file1  file2  file5  file6  file7
$ ls file[1-35-6a-c]
file1  file2  file3  file5  file6  filea  fileb  filec
```

Os colchetes também podem ser usados para encontrar um conjunto específico de caracteres correspondentes.

```
$ ls file[1a5]
file1  file5  filea
```

Também usamos o caractere `^` como o primeiro caractere para encontrar todas as correspondências, exceto determinados caracteres.

```
$ ls file[^a]
file1  file2  file3  file4  file5  file6  file7  fileb  filec
```

A última coisa que abordaremos nesta lição são as classes de caracteres. Para buscar correspondências em uma classe de caracteres, use `[:classname:]`. Por exemplo, para usar a classe digit, que corresponde aos números, faríamos algo assim:

```
$ ls file[[:digit:]]
file1  file2  file3  file4  file5  file6  file7
$ touch file1a file11
$ ls file[[:digit:]a]
file1  file2  file3  file4  file5  file6  file7  filea
$ ls file[[:digit:]]a
file1a
```

O glob `file[[:digit:]a]` corresponde a `file` seguido por um dígito ou `a`.

As classes de caracteres suportadas dependem da sua localidade atual. O POSIX requer as seguintes classes de caracteres para todas as localidades:

`[:alnum:]`
: Letras e números.

`[:alpha:]`
: Maiúsculas ou minúsculas.

`[:blank:]`
: Espaços e tabulações.

`[:cntrl:]`
: Caracteres de controle, como backspace, bell, confirmação negativa, escape.

`[:digit:]`
Numerais (`0123456789`).

`[:graph:]`
Caracteres gráficos (todos os caracteres, exceto `ctrl` e o caractere de espaço)

`[:lower:]`
Letras minúsculas (`a-z`).

`[:print:]`
Caracteres imprimíveis (`alnum`, `punct` e o caractere de espaço).

`[:punct:]`
Caracteres de pontuação, ou seja, `!`, `&`, `"`.

`[:space:]`
Caracteres de espaço em branco, por exemplo, tabulações, espaços, novas linhas.

`[:upper:]`
Letras maiúsculas (`A-Z`).

`[:xdigit:]`
Números hexadecimais (geralmente `0123456789abcdefABCDEF`).

Exercícios Guiados

1. Dadas as informações a seguir, selecione os diretórios que seriam criados com o comando `mkdir -p /tmp/outfiles/text/today /tmp/infiles/text/today`

```
$ pwd
/tmp
$ find
.
./outfiles
./outfiles/text
```

`/tmp`	
`/tmp/outfiles`	
`/tmp/outfiles/text`	
`/tmp/outfiles/text/today`	
`/tmp/infiles`	
`/tmp/infiles/text`	
`/tmp/infiles/text/today`	

2. O que `-v` faz em `mkdir`, `rm` e `cp`?

3. O que acontece se você acidentalmente tentar copiar três arquivos na mesma linha de comando para um arquivo que já existe em vez de um diretório?

4. O que acontece quando você usa `mv` para mover um diretório para dentro de si mesmo?

5. Como você excluiria todos os arquivos no diretório atual que começa com `old`?

6. Qual dos seguintes arquivos corresponderia a `log_[a-z]_201?_*_01.txt`?

`log_3_2017_Jan_01.txt`	
`log_+_2017_Feb_01.txt`	

`log_b_2007_Mar_01.txt`	
`log_f_201A_Wednesday_01.txt`	

7. Crie alguns globs que correspondam à seguinte lista de nomes de arquivos:

```
doc100
doc200
doc301
doc401
```

Exercícios Exploratórios

1. Use a página do manual de `cp` para descobrir como fazer uma cópia de um arquivo com as permissões e a data de modificação correspondentes às do original.

2. O que o comando `rmdir -p` faz? Experimente-o e explique como ele difere de `rm -r`.

3. NÃO EXECUTE ESTE COMANDO: O que você acha que `rm -ri / *` pode fazer? (É SÉRIO, NÃO TENTE EXECUTÁ-LO!)

4. Além de usar `-i`, é possível impedir que o `mv` sobrescreva os arquivos de destino?

5. Explique o comando `cp -u`.

Resumo

O ambiente de linha de comando do Linux fornece ferramentas para gerenciar arquivos. Algumas das mais usadas são `cp`, `mv`, `mkdir`, `rm` e `rmdir`. Essas ferramentas, combinadas com globs, permitem realizar tarefas com mais rapidez.

Muitos comandos têm uma opção `-i`, que solicita uma confirmação antes de fazer qualquer coisa. Essa confirmação ajuda a evitar muitos aborrecimentos se você digitar algo errado.

Muitos comandos incluem uma opção `-r`. A opção `-r` normalmente significa recursão. Em matemática e ciência da computação, uma função recursiva é uma função que utiliza a si mesma em sua definição. Quando se trata de ferramentas de linha de comando, geralmente significa aplicar o comando a um diretório e tudo o que está contido nele.

Comandos usados nesta lição:

`cat`
: Lê e exibe o conteúdo de um arquivo.

`cp`
: Copia arquivos ou diretórios.

`echo`
: Exibe um conjunto de caracteres.

`find`
: Atravessa uma árvore do sistema de arquivos e procura por arquivos que correspondam a um conjunto específico de critérios.

`ls`
: Mostra as propriedades dos arquivos e diretórios e lista o conteúdo de um diretório.

`mkdir`
: Cria novos diretórios.

`mv`
: Move ou renomeia arquivos ou diretórios.

`pwd`
: Exibe o diretório de trabalho atual..

`rm`

Exclui arquivos ou diretórios.

`rmdir`

Exclui diretórios.

`touch`

Cria novos arquivos vazios ou atualiza a data de modificação de um arquivo existente.

Respostas aos Exercícios Guiados

1. Dadas as informações a seguir, selecione os diretórios que seriam criados com o comando `mkdir -p /tmp/outfiles/text/today /tmp/infiles/text/today`

```
$ pwd
/tmp
$ find
.
./outfiles
./outfiles/text
```

   Os diretórios marcados seriam criados. Os diretórios `/tmp`, `/tmp/outfiles` e `/tmp/outfiles/text` já existem, por isso o `mkdir` os ignora.

`/tmp`	
`/tmp/outfiles`	
`/tmp/outfiles/text`	
`/tmp/outfiles/text/today`	X
`/tmp/infiles`	X
`/tmp/infiles/text`	X
`/tmp/infiles/text/today`	X

2. O que `-v` faz em `mkdir`, `rm` e `cp`?

   Normalmente, `-v` ativa a saída detalhada. Ele faz com que os programas respectivos exibam o que estão fazendo no momento em que o fazem:

```
$ rm -v a b
removed 'a'
removed 'b'
$ mv -v a b
'a' -> 'b'
$ cp -v b c
'b' -> 'c'
```

3. O que acontece se você acidentalmente tentar copiar três arquivos na mesma linha de comando para um arquivo que já existe em vez de um diretório?

`cp` se recusará a fazer qualquer coisa e exibirá uma mensagem de erro:

```
$ touch a b c d
$ cp a b c d
cp: target 'd' is not a directory
```

4. O que acontece quando você usa `mv` para mover um diretório para dentro de si mesmo?

   Aparece uma mensagem de erro dizendo que `mv` não pode fazer isso.

```
$ mv a a
mv: cannot move 'a' to a subdirectory of itself, 'a/a'
```

5. Como você excluiria todos os arquivos no diretório atual que começa com `old`?

   Seria preciso usar o glob `old*` com `rm`:

```
$ rm old*
```

6. Qual dos seguintes arquivos corresponderia a `log_[a-z]_201?_*_01.txt`?

`log_3_2017_Jan_01.txt`	
`log_+_2017_Feb_01.txt`	
`log_b_2007_Mar_01.txt`	
`log_f_201A_Wednesday_01.txt`	X

```
$ ls log_[a-z]_201?_*_01.txt
log_f_201A_Wednesday_01.txt
```

   `log_[a-z]` busca por `log_` seguido por qualquer letra minúscula, então `log_f_201A_Wednesday_01.txt` e `log_b_2007_Mar_01.txt` corresponderiam. `_201?` busca qualquer caractere único, então apenas `log_f_201A_Wednesday_01.txt` corresponderia. Finalmente, `*_01.txt` busca qualquer coisa que termine com `_01.txt`, o que corresponderia à opção restante.

7. Crie alguns globs que correspondam à seguinte lista de nomes de arquivos:

```
doc100
```

```
doc200
doc301
doc401
```

Existem várias soluções. Eis algumas delas:

```
doc*
doc[1-4]*
doc?0?
doc[1-4]0?
```

Respostas aos Exercícios Exploratórios

1. Use a página do manual de `cp` para descobrir como fazer uma cópia de um arquivo com as permissões e a data de modificação correspondentes às do original.

   Usaríamos nesse caso a opção `-p`. Segundo a página man:

   ```
   $ man cp
   -p     same as --preserve=mode,ownership,timestamps
   --preserve[=ATTR_LIST]
                preserve the specified attributes (default: mode,ownership,time-
                stamps), if  possible  additional  attributes:  context,  links,
                xattr, all
   ```

2. O que o comando `rmdir -p` faz? Experimente-o e explique como ele difere de `rm -r`.

   Ele faz com que `rmdir` se comporte de maneira semelhante a `mkdir -p`. Se for passado em uma árvore de diretórios vazios, ele removerá todos eles.

   ```
   $ find
   .
   ./a
   ./a/b
   ./a/b/c
   $ rmdir -p a/b/c
   $ ls
   ```

3. NÃO EXECUTE ESTE COMANDO: O que você acha que `rm -ri / *` pode fazer? (É SÉRIO, NÃO TENTE EXECUTÁ-LO!)

   Ele remove todos os arquivos e diretórios com permissão de escrita em sua conta de usuário. Isso inclui quaisquer sistemas de arquivos de rede.

4. Além de usar `-i`, é possível impedir que o `mv` sobrescreva os arquivos de destino?

   Sim, a opção `-n` ou `--no-clobber` impede que `mv` sobrescreva arquivos.

   ```
   $ cat a
   a
   $ cat b
   b
   ```

```
$ mv -n a b
$ cat b
b
```

5. Explique `cp -u`.

   A opção `-u` faz com que `cp` só copie um arquivo se o destino estiver ausente ou for mais antigo que o arquivo de origem.

```
$ ls -l
total 24K
drwxr-xr-x 123 emma student  12K Feb  2 05:34 ..
drwxr-xr-x   2 emma student 4.0K Feb  2 06:56 .
-rw-r--r--   1 emma student    2 Feb  2 06:56 a
-rw-r--r--   1 emma student    2 Feb  2 07:00 b
$ cat a
a
$ cat b
b
$ cp -u a b
$ cat b
b
$ cp -u a c
$ ls -l
total 12
-rw-r--r-- 1 emma student 2 Feb  2 06:56 a
-rw-r--r-- 1 emma student 2 Feb  2 07:00 b
-rw-r--r-- 1 emma student 2 Feb  2 07:00 c
```

Linux Professional Institute

Tópico 3: O Poder da Linha de Comando

3.1 Empacotando arquivos na linha de comando

Referência ao LPI objectivo

Linux Essentials version 1.6, Exam 010, Objective 3.1

Peso

2

Áreas chave de conhecimento

- Arquivos, diretórios
- Pacotes, compressão

Segue uma lista parcial de arquivos, termos e utilitários utilizados

- `tar`
- Opções comuns do `tar`
- `gzip`, `bzip2`, `xz`
- `zip`, `unzip`

Linux Professional Institute

3.1 Lição 1

Certificate:	Linux Essentials
Version:	1.6
Topic:	3 O poder da linha de comando
Objective:	3.1 Como comprimir arquivos com a linha de comando
Lesson:	1 of 1

Introdução

A compressão é usada para reduzir a quantidade de espaço que um conjunto específico de dados consome. Em geral, recorremos a ela para reduzir a quantidade de espaço necessária para armazenar um arquivo. Outro uso comum é reduzir a quantidade de dados enviados através de uma conexão de rede.

A compressão substitui os padrões repetitivos presentes nos dados. Imagine um romance. Algumas palavras são extremamente comuns, mas têm mais de um caractere, como a palavra “mas”. É possível reduzir bastante o tamanho do romance substituindo essas palavras comuns com vários caracteres por outras formadas por um único caractere - por exemplo, substituindo “mas” por uma letra grega que não é usada em nenhuma outra parte do texto. Os algoritmos de compressão de dados funcionam de maneira semelhante, mas mais complexa.

Há duas variedades de compressão: *sem perda* e *com perda*. Os arquivos comprimidos com um algoritmo sem perda podem ser descomprimidos em sua forma original. Já os dados comprimidos com um algoritmo com perda não podem ser recuperados. Os algoritmos com perda são frequentemente usados para imagens, vídeos e áudios em que a perda de qualidade é

imperceptível para os seres humanos, irrelevante para o contexto, ou em que a economia de espaço e de taxa de transferência de rede compensam a perda.

Usamos ferramentas de empacotamento para agrupar arquivos e diretórios em um único arquivo. Alguns usos comuns são backups, empacotamento de código-fonte de software e retenção de dados.

O empacotamento e a compressão são comumente usados juntos. Algumas ferramentas de empacotamento também comprimem seu conteúdo por padrão. Outras trazem a opção de comprimir o conteúdo. Algumas ferramentas de empacotamento devem ser usadas em conjunto com ferramentas de compressão independentes, caso se deseje comprimir o conteúdo.

A ferramenta mais comum para empacotar arquivos em sistemas Linux é o `tar`. A maioria das distribuições Linux é fornecida com a versão GNU do `tar`, e por isso a usaremos nesta lição. O `tar`, por si só, gerencia o empacotamento de arquivos, mas não os comprime.

Existem muitas ferramentas de compressão disponíveis no Linux. Dentre as sem perdas, encontramos comumente `bzip2`, `gzip` e `xz`. As três estão presentes na maioria dos sistemas. Em sistemas antigos ou muito minimalistas, o `xz` ou o `bzip` talvez não estejam instalados. Se você se tornar um usuário regular do Linux, provavelmente vai encontrar arquivos comprimidos com todas as três. Elas usam algoritmos diferentes; portanto, um arquivo comprimido com uma ferramenta não pode ser descomprimido por outra. As ferramentas de compressão exigem uma escolha: se você deseja uma alta taxa de compressão, levará mais tempo para comprimir e descomprimir o arquivo. Isso ocorre porque uma compressão mais alta demanda mais trabalho para encontrar padrões mais complexos. Todas essas ferramentas comprimem dados, mas não podem criar pacotes contendo vários arquivos.

Ferramentas exclusivas para compressão não costumam estar disponíveis nos sistemas Windows. As ferramentas de empacotamento e compressão do Windows geralmente são agrupadas. Lembre-se disso caso possua sistemas Linux e Windows que precisam compartilhar arquivos.

Os sistemas Linux também trazem ferramentas para lidar com os arquivos `.zip`, comumente usados no sistema Windows. Elas se chamam `zip` e `unzip`. Essas ferramentas não vêm instaladas por padrão em todos os sistemas; se você precisar delas, poderá ser necessário instalá-las. Felizmente, elas costumam estar presentes nos repositórios de pacotes das distribuições.

Ferramentas de compressão

A quantidade de espaço em disco economizada pela compressão de arquivos depende de alguns fatores: a natureza dos dados que estão sendo comprimidos, o algoritmo usado para comprimir os dados e o nível de compressão. Nem todos os algoritmos suportam níveis de compressão

diferentes.

Vamos começar preparando alguns arquivos de teste para comprimir:

```
$ mkdir ~/linux_essentials-3.1
$ cd ~/linux_essentials-3.1
$ mkdir compression archiving
$ cd compression
$ cat /etc/* > bigfile 2> /dev/null
```

Agora, criamos três cópias desse arquivo:

```
$ cp bigfile bigfile2
$ cp bigfile bigfile3
$ cp bigfile bigfile4
$ ls -lh
total 2.8M
-rw-r--r-- 1 emma emma 712K Jun 23 08:08 bigfile
-rw-r--r-- 1 emma emma 712K Jun 23 08:08 bigfile2
-rw-r--r-- 1 emma emma 712K Jun 23 08:08 bigfile3
-rw-r--r-- 1 emma emma 712K Jun 23 08:08 bigfile4
```

Em seguida vamos comprimir os arquivos com cada uma das ferramentas de compressão mencionadas:

```
$ bzip2 bigfile2
$ gzip bigfile3
$ xz bigfile4
$ ls -lh
total 1.2M
-rw-r--r-- 1 emma emma 712K Jun 23 08:08 bigfile
-rw-r--r-- 1 emma emma 170K Jun 23 08:08 bigfile2.bz2
-rw-r--r-- 1 emma emma 179K Jun 23 08:08 bigfile3.gz
-rw-r--r-- 1 emma emma 144K Jun 23 08:08 bigfile4.xz
```

Compare o tamanho dos arquivos comprimidos com o arquivo descomprimido chamado `bigfile`. Observe também como as ferramentas de compressão adicionaram extensões aos nomes dos arquivos e removeram os arquivos não comprimidos.

Use `bunzip2`, `gunzip` ou `unxz` para descomprimir os arquivos:

```
$ bunzip2 bigfile2.bz2
$ gunzip bigfile3.gz
$ unxz bigfile4.xz
$ ls -lh
total 2.8M
-rw-r--r-- 1 emma emma 712K Jun 23 08:20 bigfile
-rw-r--r-- 1 emma emma 712K Jun 23 08:20 bigfile2
-rw-r--r-- 1 emma emma 712K Jun 23 08:20 bigfile3
-rw-r--r-- 1 emma emma 712K Jun 23 08:20 bigfile4
```

Observe que, desta vez, o arquivo comprimido foi excluído depois de descomprimido.

Certas ferramentas permitem diferentes níveis de compressão. Um nível de compressão mais alto geralmente requer mais memória e ciclos de CPU, mas resulta em um arquivo comprimido menor. O oposto é verdadeiro para um nível mais baixo. Veja abaixo uma demonstração com `xz` e `gzip`:

```
$ cp bigfile bigfile-gz1
$ cp bigfile bigfile-gz9
$ gzip -1 bigfile-gz1
$ gzip -9 bigfile-gz9
$ cp bigfile bigfile-xz1
$ cp bigfile bigfile-xz9
$ xz -1 bigfile bigfile-xz1
$ xz -9 bigfile bigfile-xz9
$ ls -lh bigfile bigfile-* *
total 3.5M
-rw-r--r-- 1 emma emma 712K Jun 23 08:08 bigfile
-rw-r--r-- 1 emma emma 205K Jun 23 13:14 bigfile-gz1.gz
-rw-r--r-- 1 emma emma 178K Jun 23 13:14 bigfile-gz9.gz
-rw-r--r-- 1 emma emma 156K Jun 23 08:08 bigfile-xz1.xz
-rw-r--r-- 1 emma emma 143K Jun 23 08:08 bigfile-xz9.xz
```

Não é necessário descomprimir um arquivo toda vez que ele for usado. As ferramentas de compressão geralmente vêm com versões especiais de ferramentas comuns usadas para ler arquivos de texto. Por exemplo, o `gzip` tem uma versão de `cat`, `grep`, `diff`, `less`, `more` e algumas outras. No `gzip`, as ferramentas são prefixadas com um `z`, enquanto o prefixo `bz` existe para o `bzip2` e `xz` para o `xz`. Veja abaixo um exemplo do uso de `zcat` para ler um arquivo compactado com `gzip`:

```
$ cp /etc/hosts ./
$ gzip hosts
```

```
$ zcat hosts.gz
127.0.0.1   localhost

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Ferramentas de empacotamento

O programa `tar` é provavelmente a ferramenta de empacotamento mais usada nos sistemas Linux. Caso você esteja se perguntando por que ela tem esse nome, trata-se de uma abreviação de "tape archive" (arquivo de fita). Os arquivos criados com o `tar` são apelidados de *tar balls*. É muito comum que aplicativos distribuídos na forma de código-fonte estejam em tar balls.

A versão GNU do `tar` incluída nas distribuições Linux tem muitas opções. Esta lição abordará o subconjunto mais usado.

Vamos começar criando um pacote com os arquivos usados para a compressão:

```
$ cd ~/linux_essentials-3.1
$ tar cf archiving/3.1.tar compression
```

A opção `c` instrui o `tar` a criar um novo arquivo e a opção `f` é o nome do arquivo a ser criado. O argumento imediatamente após as opções sempre será o nome do arquivo no qual trabalhar. O restante dos argumentos são os caminhos para quaisquer arquivos ou diretórios que você deseje adicionar, listar ou extrair do arquivo. No exemplo, adicionamos o diretório `compression` e todo o seu conteúdo ao pacote.

Para visualizar o conteúdo de um tar ball, use a opção `t` do `tar`:

```
$ tar -tf 3.1.tar
compression/
compression/bigfile-xz1.xz
compression/bigfile-gz9.gz
compression/hosts.gz
compression/bigfile2
compression/bigfile
compression/bigfile-gz1.gz
compression/bigfile-xz9.xz
compression/bigfile3
```

```
compression/bigfile4
```

Note que as opções são precedidas por `-`. Ao contrário da maioria dos programas, com o `tar` o `-` não é necessário ao especificar opções, embora não cause nenhum dano se for usado.

NOTE Use a opção `-v` para permitir que o `tar` exiba o nome dos arquivos nos quais opera ao criar ou extrair um pacote.

Agora vamos extrair o arquivo:

```
$ cd ~/linux_essentials-3.1/archiving
$ ls
3.1.tar
$ tar xf 3.1.tar
$ ls
3.1.tar  compression
```

Suponha que você precise apenas de um arquivo que esteja no pacote. Se for esse o caso, você pode especificá-lo após o nome de arquivo do pacote. É possível especificar vários arquivos, se necessário:

```
$ cd ~/linux_essentials-3.1/archiving
$ rm -rf compression
$ ls
3.1.tar
$ tar xvf 3.1.tar compression/hosts.gz
compression/
compression/bigfile-xz1.xz
compression/bigfile-gz9.gz
compression/hosts.gz
compression/bigfile2
compression/bigfile
compression/bigfile-gz1.gz
compression/bigfile-xz9.xz
compression/bigfile3
compression/bigfile4
$ ls
3.1.tar  compression
$ ls compression
hosts.gz
```

Com exceção dos caminhos absolutos (caminhos começando com `/`), os arquivos `tar` preservam todo o caminho dos arquivos quando eles são criados. Como o arquivo `3.1.tar` foi criado com um único diretório, esse diretório será criado em relação ao seu diretório de trabalho atual quando for extraído. Para que isso fique mais claro, eis outro exemplo:

```
$ cd ~/linux_essentials-3.1/archiving
$ rm -rf compression
$ cd ../compression
$ tar cf ../tar/3.1-nodir.tar *
$ cd ../archiving
$ mkdir untar
$ cd untar
$ tar -xf ../3.1-nodir.tar
$ ls
bigfile   bigfile3  bigfile-gz1.gz  bigfile-xz1.xz  hosts.gz
bigfile2  bigfile4  bigfile-gz9.gz  bigfile-xz9.xz
```

TIP Caso queira usar o caminho absoluto em um arquivo `tar`, é necessário empregar a opção `P`. Esteja ciente de que isso pode sobrescrever arquivos importantes e causar erros no sistema.

O programa `tar` também pode gerenciar a compressão e descompressão de arquivos em tempo real. Para isso, o `tar` chama uma das ferramentas de compressão discutidas anteriormente nesta seção. Basta adicionar a opção apropriada ao algoritmo de compressão. As mais usadas são `j`, `J` e `z` para `bzip2`, `xz` e `gzip`, respectivamente. Eis alguns exemplos usando os algoritmos mencionados acima:

```
$ cd ~/linux_essentials-3.1/compression
$ ls
bigfile   bigfile3  bigfile-gz1.gz  bigfile-xz1.xz  hosts.gz
bigfile2  bigfile4  bigfile-gz9.gz  bigfile-xz9.xz
$ tar -czf gzip.tar.gz bigfile bigfile2 bigfile3
$ tar -cjf bzip2.tar.bz2 bigfile bigfile2 bigfile3
$ tar -cJf xz.tar.xz bigfile bigfile2 bigfile3
$ ls -l | grep tar
-rw-r--r-- 1 emma emma  450202 Jun 27 05:56 bzip2.tar.bz2
-rw-r--r-- 1 emma emma  548656 Jun 27 05:55 gzip.tar.gz
-rw-r--r-- 1 emma emma  147068 Jun 27 05:56 xz.tar.xz
```

Observe como, no exemplo, os arquivos `.tar` têm tamanhos diferentes. Isso mostra que foram compactados com sucesso. Ao criar arquivos compactados `.tar`, sempre se deve adicionar uma

segunda extensão de arquivo indicando o algoritmo usado. Elas são `.xz`, `.bz` e `.gz` para `xz`, `bzip2` e `gzip`, respectivamente. Às vezes, usam-se extensões encurtadas, como `.tgz`.

É possível adicionar arquivos a pacotes tar não comprimidos já existentes. Para isso, usamos a opção `u`. Se você tentar adicionar arquivos a um arquivo comprimido, aparecerá uma mensagem de erro.

```
$ cd ~/linux_essentials-3.1/compression
$ ls
bigfile   bigfile3  bigfile-gz1.gz  bigfile-xz1.xz  bzip2.tar.bz2  hosts.gz
bigfile2  bigfile4  bigfile-gz9.gz  bigfile-xz9.xz  gzip.tar.gz    xz.tar.xz
$ tar cf plain.tar bigfile bigfile2 bigfile3
$ tar tf plain.tar
bigfile
bigfile2
bigfile3
$ tar uf plain.tar bigfile4
$ tar tf plain.tar
bigfile
bigfile2
bigfile3
bigfile4
$ tar uzf gzip.tar.gz bigfile4
tar: Cannot update compressed archives
Try 'tar --help' or 'tar --usage' for more information.
```

Gerenciando arquivos ZIP

As máquinas Windows geralmente não têm aplicativos para lidar com tar balls ou muitas das ferramentas de compressão comumente encontradas nos sistemas Linux. Se você precisar interagir com sistemas Windows, é aconselhável usar arquivos ZIP. Um arquivo ZIP é um pacote semelhante a um arquivo `tar` compactado.

Os programas `zip` e `unzip` servem para trabalhar com arquivos ZIP em sistemas Linux. O exemplo abaixo mostra todo o necessário para você começar a usá-los. Primeiro, criamos um conjunto de arquivos:

```
$ cd ~/linux_essentials-3.1
$ mkdir zip
$ cd zip/
$ mkdir dir
```

```
$ touch dir/file1 dir/file2
```

Em seguida, usamos `zip` para comprimir esses arquivos em um arquivo ZIP:

```
$ zip -r zipfile.zip dir
  adding: dir/ (stored 0%)
  adding: dir/file1 (stored 0%)
  adding: dir/file2 (stored 0%)
$ rm -rf dir
```

Por fim, descompactamos o arquivo ZIP novamente:

```
$ ls
zipfile.zip
$ unzip zipfile.zip
Archive:  zipfile.zip
   creating: dir/
 extracting: dir/file1
 extracting: dir/file2
$ find
.
./zipfile.zip
./dir
./dir/file1
./dir/file2
```

Ao adicionar diretórios aos arquivos ZIP, a opção `-r` faz com que o `zip` inclua o conteúdo desses diretórios. Sem ele, teríamos um diretório vazio no arquivo ZIP.

Exercícios Guiados

1. Com base nas extensões, quais das seguintes ferramentas foram usadas para criar estes arquivos?

Nome do Arquivo	`tar`	`gzip`	`bzip2`	`xz`
`archive.tar`				
`archive.tgz`				
`archive.tar.xz`				

2. Com base nas extensões, quais destes arquivos estão empacotados e quais estão comprimidos?

Nome do Arquivo	**Empacotado**	**Comprimido**
`file.tar`		
`file.tar.bz2`		
`file.zip`		
`file.xz`		

3. Como se adiciona um arquivo a um arquivo `tar` comprimido com `gzip`?

4. Qual opção de `tar` instrui o `tar` a incluir o caractere inicial `/` nos caminhos absolutos?

5. O `zip` suporta diferentes níveis de compressão?

Exercícios Exploratórios

1. Ao extrair arquivos, o `tar` suporta globs na lista de arquivos?

2. Como garantir que um arquivo descomprimido fique igual ao arquivo antes da compressão?

3. O que acontece quando tentamos extrair de um pacote `tar` um arquivo que já existe em seu sistema de arquivos?

4. Como extrair o arquivo `archive.tgz` sem usar a opção `tar z`?

Resumo

Os sistemas Linux oferecem diversas ferramentas de compressão e empacotamento. Esta lição abordou as mais comuns. A ferramenta de empacotamento mais comum é o `tar`. Se for necessário interagir com sistemas Windows, `zip` e `unzip` ajudam a criar e extrair arquivos ZIP.

O comando `tar` tem algumas opções que vale a pena memorizar. Elas são: `x` para extrair, `c` para criar, `t` para visualizar o conteúdo e `u` para adicionar ou substituir arquivos. A opção `v` lista os arquivos que são processados pelo `tar` ao criar ou extrair um arquivo.

O repositório típico de uma distribuição Linux inclui muitas ferramentas de compressão. As mais comuns são `gzip`, `bzip2` e `xz`. Os algoritmos de compressão geralmente oferecem suporte a diferentes níveis de compressão para otimizar a velocidade ou o tamanho do arquivo. Os arquivos podem ser descomprimidos com `gunzip`, `bunzip2` e `unxz`.

As ferramentas de compressão geralmente têm programas que se comportam como ferramentas comuns para arquivos de texto, com a diferença de que funcionam em arquivos comprimidos. Dentre eles temos o `zcat`, `bzcat` e `xzcat`. As ferramentas de compressão costumam vir com programas com as funcionalidades `grep`, `more`, `less`, `diff` e `cmp`.

Comandos usados nos exercícios:

`bunzip2`
: Descomprime um arquivo comprimido `bzip2`.

`bzcat`
: Exibe o conteúdo de um arquivo comprimido `bzip`.

`bzip2`
: Comprime arquivos usando o algoritmo e o formato `bzip2`.

`gunzip`
: Descomprime um arquivo comprimido `gzip`.

`gzip`
: Comprime arquivos usando o algoritmo e o formato `gzip`.

`tar`
: Cria, atualiza, lista e extrai pacotes `tar`.

`unxz`

Descomprime um arquivo comprimido `xz`.

`unzip`

Descomprime e extrai conteúdo de um arquivo ZIP.

`xz` Comprime arquivos usando o algoritmo e o formato `xz`.

`zcat`

Exibe o conteúdo de um arquivo comprimido `gzip`.

`zip`

Cria e comprime pacotes ZIP.

Respostas aos Exercícios Guiados

1. Com base nas extensões, quais das seguintes ferramentas foram usadas para criar estes arquivos?

Nome do Arquivo	`tar`	`gzip`	`bzip2`	`xz`
`archive.tar`	X			
`archive.tgz`	X	X		
`archive.tar.xz`	X			X

2. Com base nas extensões, quais destes arquivos estão empacotados e quais estão comprimidos?

Nome do Arquivo	**Empacotado**	**Comprimido**
`file.tar`	X	
`file.tar.bz2`	X	X
`file.zip`	X	X
`file.xz`		X

3. Como se adiciona um arquivo a um arquivo `tar` comprimido com `gzip`?

   Descomprimimos o arquivo com `gunzip`, adicionamos o arquivo com `tar uf` e em seguida o comprimimos com `gzip`

4. Qual opção de `tar` instrui o `tar` a incluir o caractere inicial `/` nos caminhos absolutos?

   A opção `-P`. Segundo a página man:

```
-P, --absolute-names
       Don't strip leading slashes from file names when creating  archives
```

5. O `zip` suporta diferentes níveis de compressão?

   Sim. Usaríamos `-#`, substituindo `#` por um número de 0-9. Segundo a página man:

```
-#
(-0, -1, -2, -3, -4, -5, -6, -7, -8, -9)
       Regula a velocidade da compressão usando o dígito especificado #, onde -0 indica
```

```
sem compressão (armazena todos os arquivos), -1 indica a velocidade de compressão mais
rápida (menos compressão) e -9 indica a velocidade de compressão mais lenta (compressão
ideal, ignora a lista de sufixos). O nível de compressão padrão é -6.

       Embora ainda esteja sendo trabalhada, a intenção é que essa configuração controle
a velocidade de compressão para todos os métodos de compressão. Atualmente, apenas a
deflação é controlada.
```

Respostas aos Exercícios Exploratórios

1. Ao extrair arquivos, o `tar` suporta globs na lista de arquivos?

   Sim, com a opção `--wildcards`. `--wildcards` deve ser colocado logo após o arquivo `tar` ao se usar o estilo de opções sem hífen. Por exemplo:

   ```
   $ tar xf tarfile.tar --wildcards dir/file*
   $ tar --wildcards -xf tarfile.tar dir/file*
   ```

2. Como garantir que um arquivo descomprimido fique igual ao arquivo antes da compressão?

   Não é preciso fazer nada com as ferramentas mostradas nesta lição. As três incluem somas de controle no formato de arquivo, que são verificadas quando eles são descomprimidos.

3. O que acontece quando tentamos extrair de um pacote `tar` um arquivo que já existe em seu sistema de arquivos?

   O arquivo no sistema de arquivos é sobrescrito com a versão que está no arquivo `tar`.

4. Como extrair o arquivo `archive.tgz` sem usar a opção `tar z`?

   Seria preciso descomprimi-lo antes com o `gunzip`.

   ```
   $ gunzip archive.tgz
   $ tar xf archive.tar
   ```

3.2 Pesquisando e extraindo dados de arquivos

Referência ao LPI objectivo

Linux Essentials version 1.6, Exam 010, Objective 3.2

Peso

3

Áreas chave de conhecimento

- Pipes da linha de comando
- Redirecionamento I/O
- Expressões regulares básicas utilizando `.`, `[ ]`, `*`, e `?`

Segue uma lista parcial de arquivos, termos e utilitários utilizados

- `grep`
- `less`
- `cat`, `head`, `tail`
- `sort`
- `cut`
- `wc`

3.2 Lesson 1

Certificate:	Linux Essentials
Version:	1.6
Topic:	3 O poder da linha de comando
Objective:	3.2 Como buscar e extrair dados de arquivos
Lesson:	1 of 2

Introdução

Nesta lição, vamos nos concentrar no redirecionamento ou transmissão de informações de uma fonte para outra com a ajuda de ferramentas específicas. A linha de comando do Linux redireciona as informações através de canais padrão específicos. O teclado é considerado a entrada padrão (*stdin* ou canal 0) do comando, e a tela, a saída padrão (*stdout* ou canal 1). Há também um terceiro canal destinado a redirecionar a saída de erro (*stderr* ou canal 2) de um comando ou as mensagens de erro de um programa. A entrada e/ou saída podem ser redirecionadas.

Ao executar um comando, às vezes queremos transmitir determinadas informações a ele ou redirecionar a saída para um arquivo específico. Cada uma dessas funcionalidades será discutida nas duas seções a seguir.

Redirecionamento de E/S

O redirecionamento de E/S permite que o usuário redirecione informações de ou para um comando usando um arquivo de texto. Como explicado anteriormente, a entrada, a saída e a saída de erro padrão podem ser redirecionadas, e as informações, obtidas de arquivos de texto.

Redirecionando a saída padrão

Para redirecionar a saída padrão para um arquivo em vez da tela, devemos usar o operador `>` seguido pelo nome do arquivo. Se o arquivo não existir, um novo será criado; caso contrário, a informação vai sobrescrever o arquivo existente.

Para ver o conteúdo do arquivo que acabamos de criar, podemos usar o comando `cat`. Por padrão, esse comando exibe o conteúdo de um arquivo na tela. Consulte a página do manual para saber mais sobre suas funcionalidades.

O exemplo abaixo demonstra a funcionalidade do operador. Na primeira instância, um novo arquivo é criado contendo o texto “Hello World!”:

```
$ echo "Hello World!" > text
$ cat text
Hello World!
```

Na segunda invocação, o mesmo arquivo é sobrescrito com o novo texto:

```
$ echo "Hello!" > text
$ cat text
Hello!
```

Se quisermos adicionar novas informações ao final do arquivo, usamos o operador `>>`. Esse operador também cria um novo arquivo caso não encontre um arquivo existente.

O primeiro exemplo mostra a adição do texto. Como vemos, o novo texto é adicionado na linha seguinte:

```
$ echo "Hello to you too!" >> text
$ cat text
Hello!
Hello to you too!
```

O segundo exemplo demonstra que um novo arquivo será criado:

```
$ echo "Hello to you too!" >> text2
$ cat text2
Hello to you too!
```

Redirecionando o erro padrão

Para redirecionar apenas as mensagens de erro, o administrador do sistema precisará usar o operador `2>` seguido pelo nome do arquivo no qual os erros serão escritos. Se o arquivo não existir, um novo será criado; caso contrário, o arquivo será sobrescrito.

Como explicado, o canal para redirecionar o erro padrão é o *canal 2*. Ao redirecionar o erro padrão, o canal deve ser especificado, ao contrário da outra saída padrão, em que o *canal 1* já está definido a princípio. O comando no exemplo a seguir procura por um arquivo ou diretório chamado `games` e escreve apenas o erro no arquivo `text-error`, enquanto exibe a saída padrão na tela:

```
$ find /usr games 2> text-error
/usr
/usr/share
/usr/share/misc
---------Omitted output----------
/usr/lib/libmagic.so.1.0.0
/usr/lib/libdns.so.81
/usr/games
$ cat text-error
find: `games': No such file or directory
```

NOTE Para saber mais sobre o comando `find`, consulte a página de manual correspondente.

O comando do exemplo a seguir será executado sem erros e, portanto, nenhuma informação será gravada no arquivo `text-error`:

```
$ sort /etc/passwd 2> text-error
$ cat text-error
```

Além da saída padrão, o erro padrão também pode ser incluído em um arquivo com o operador `2>>`. Isso adicionará o novo erro no final do arquivo. Se o arquivo não existir, um novo será criado. O primeiro exemplo mostra a adição das novas informações ao arquivo. O segundo exemplo mostra que o comando cria um novo arquivo quando um arquivo existente com o mesmo nome não pode ser encontrado:

```
$ sort /etc 2>> text-error
$ cat text-error
```

```
sort: read failed: /etc: Is a directory
```

```
$ sort /etc/shadow 2>> text-error2
$ cat text-error2
sort: open failed: /etc/shadow: Permission denied
```

Usando esse tipo de redirecionamento, apenas as mensagens de erro serão redirecionadas para o arquivo; a saída normal será escrita na tela ou passará pela saída padrão ou *stdout*.

Existe um arquivo em particular que tecnicamente é um *bit bucket* (um arquivo que aceita entrada e não faz nada com ela): `/dev/null`. É possível redirecionar para lá qualquer informação irrelevante que não precise ser exibida ou redirecionada para um arquivo importante, como mostrado no exemplo abaixo:

```
$ sort /etc 2> /dev/null
```

Redirecionando a entrada padrão

Esse tipo de redirecionamento é usado para inserir dados em um comando a partir de um arquivo especificado em vez de um teclado. Nesse caso, o operador `<` é usado, como no exemplo abaixo:

```
$ cat < text
Hello!
Hello to you too!
```

Em geral, redirecionamos a entrada padrão com comandos que não aceitam argumentos de arquivo. O comando `tr` é um deles. Esse comando pode ser usado para traduzir o conteúdo do arquivo modificando as palavras de maneira específica, como por exemplo a exclusão de um determinado caractere. O exemplo abaixo mostra a exclusão do caractere `l`:

```
$ tr -d "l" < text
Heo!
Heo to you too!
```

Para saber mais, consulte a página de manual do `tr`.

Here documents

Ao contrário dos redirecionamentos de saída, o operador `<<` age de maneira diferente em comparação com os outros operadores. Esse fluxo de entrada também é chamado de *here document*. O *here document* representa o bloco de código ou texto que pode ser redirecionado para o comando ou o programa interativo. Diferentes tipos de linguagens de script, como `bash`, `sh` e `csh`, podem receber entradas diretamente da linha de comando, sem usar nenhum arquivo de texto.

Como vemos no exemplo abaixo, o operador é usado para inserir dados no comando, ao passo que a palavra que vem depois não especifica o nome do arquivo. A palavra é interpretada como o delimitador da entrada e não será levada em consideração como conteúdo; portanto, o `cat` não a exibirá:

```
$ cat << hello
> hey
> ola
> hello
hey
ola
```

Consulte a página de manual do comando `cat` para saber mais.

Combinações

A primeira combinação de que trataremos combina o redirecionamento da saída padrão e o da saída de erro padrão para o mesmo arquivo. Os operadores `&>` e `&>>` são usados: `&` representa a combinação do *canal 1* e do *canal 2*. O primeiro operador substituirá o conteúdo existente no arquivo e o segundo incluirá ou adicionará as novas informações ao final do arquivo. Ambos os operadores permitem a criação de um novo arquivo se já não existir um, como nas seções anteriores:

```
$ find /usr admin &> newfile
$ cat newfile
/usr
/usr/share
/usr/share/misc
---------Omitted output----------
/usr/lib/libmagic.so.1.0.0
/usr/lib/libdns.so.81
/usr/games
```

```
find: `admin': No such file or directory
$ find /etc/calendar &>> newfile
$ cat newfile
/usr
/usr/share
/usr/share/misc
---------Omitted output----------
/usr/lib/libmagic.so.1.0.0
/usr/lib/libdns.so.81
/usr/games
find: `admin': No such file or directory
/etc/calendar
/etc/calendar/default
```

Vamos ver um exemplo que usa o comando `cut`:

```
$ cut -f 3 -d "/" newfile
$ cat newfile

share
share
share
---------Omitted output----------
lib
games
find: `admin': No such file or directory
calendar
calendar
find: `admin': No such file or directory
```

O comando `cut` corta campos especificados do arquivo de entrada usando a opção `-f`, o terceiro campo em nosso caso. Para que o comando encontre o campo, um delimitador também deve ser especificado com a opção `-d`. Em nosso caso, o delimitador será o caractere `/`.

Para saber mais sobre o comando `cut`, consulte a página man correspondente.

Pipes na linha de comando

O redirecionamento é usado principalmente para armazenar o resultado de um comando a ser processado por um comando diferente. Esse tipo de processo intermediário pode se tornar tedioso e complicado quando os dados precisam passar por múltiplos processos. Para evitar isso, podemos vincular o comando diretamente usando *pipes*. Em outras palavras, a saída do primeiro comando

se torna automaticamente a entrada do segundo comando. Essa conexão é feita usando o operador | (barra vertical):

```
$ cat /etc/passwd | less
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
:
```

No exemplo acima, o comando `less`, após o operador pipe, modifica a maneira como o arquivo é exibido. O comando `less` exibe o arquivo de texto, permitindo ao usuário rolar uma linha por vez para cima e para baixo.`less` também é usado por padrão para exibir as páginas de manual, conforme discutido nas lições anteriores.

É possível empregar várias pipes ao mesmo tempo. Os comandos intermediários que recebem uma entrada, a alteram e produzem uma saída são chamados de *filtros*. Vamos pegar o comando `ls -l` e tentar contar o número de palavras das 10 primeiras linhas da saída. Para isso, teremos de usar o comando `head` que, por padrão, exibe as 10 primeiras linhas de um arquivo, e em seguida contar as palavras usando o comando `wc`:

```
$ ls -l | head | wc -w
10
```

Como já mencionado, por padrão, o `head` exibe apenas as 10 primeiras linhas do arquivo de texto especificado. Esse comportamento pode ser modificado usando opções específicas. Verifique a página man do comando para saber mais.

Existe outro comando para exibir o final de um arquivo: `tail`. Por padrão, esse comando seleciona as últimas 10 linhas e as exibe, mas, como em `head`, esse número também pode ser modificado. Verifique a página de manual de `tail` para mais detalhes.

NOTE A opção `-f` mostra as últimas linhas de um arquivo enquanto ele está sendo atualizado. Esse recurso pode ser muito útil ao se monitorar a atividade em andamento em um arquivo como o `syslog`.

O comando `wc` (contagem de palavras) conta por padrão as linhas, palavras e bytes de um arquivo. Como mostrado no exercício, a opção `-w` faz com que o comando conte apenas as palavras dentro das linhas selecionadas. As opções mais comuns desse comando são: `-l`, que especifica que o comando deve contar apenas as linhas, e `-c`, usado para contar apenas os bytes. Mais variações e opções do comando, bem como informações sobre o `wc`, podem ser encontradas na página de

manual do comando.

Exercícios Guiados

1. Liste o conteúdo do diretório atual, incluindo o proprietário e as permissões, e redirecione a saída para um arquivo chamado `contents.txt` dentro de seu diretório inicial.

2. Ordene o conteúdo do arquivo `contents.txt` a partir do diretório atual e inclua esse conteúdo ordenado no final de um novo arquivo chamado `contents-sorted.txt`

3. Exiba as últimas 10 linhas do arquivo `/etc/passwd` e redirecione-as para um novo arquivo no diretório `Documents` do seu usuário.

4. Conte o número de palavras no arquivo `contents.txt` e inclua a saída ao final de um arquivo chamado `field2.txt` em seu diretório inicial. Você precisará usar redirecionamentos de entrada e de saída.

5. Exiba as 5 primeiras linhas do arquivo `/etc/passwd` e classifique a saída em ordem alfabética reversa.

6. Usando o arquivo `contents.txt` criado anteriormente, conte o número de caracteres das últimas 9 linhas.

7. Conte o número de arquivos chamados `test` no diretório `/usr/share` e em seus subdiretórios. Nota: cada linha da saída do comando `find` representa um arquivo.

Exercícios Exploratórios

1. Selecione o segundo campo do arquivo `contents.txt` e redirecione a saída padrão e a saída de erro para outro arquivo chamado `field1.txt`.

2. Usando o operador de redirecionamento de entrada e o comando `tr`, exclua os hífens (`-`) do arquivo `contents.txt`.

3. Qual é a maior vantagem de redirecionar apenas os erros para um arquivo?

4. Substitua todos os espaços repetidos no arquivo `contents.txt` classificado em ordem alfabética por um único espaço.

5. Em uma linha de comando, elimine os espaços repetidos (como feito no exercício anterior), selecione o nono campo e ordene-o em ordem alfabética reversa e sem distinção entre maiúsculas e minúsculas. Quantos pipes você teve de usar?

Resumo

Nesta lição, você aprendeu:

- Tipos de redirecionamento
- Como usar os operadores de redirecionamento
- Como usar pipes pra filtrar a saída dos comandos

Comandos usados nesta lição:

`cut`
Remove seções de cada linha de um arquivo.

`cat`
Exibe ou concatena arquivos.

`find`
Busca por arquivos em uma hierarquia de diretórios.

`less`
Exibe um arquivo, permitindo que o usuário role uma linha por vez.

`more`
Exibe um arquivo, uma página por vez.

`head`
Exibe as primeiras 10 linhas de um arquivo.

`tail`
Exibe as últimas 10 linhas de um arquivo.

`sort`
Classifica os arquivos.

`wc`
Conta por padrão as linhas, palavras ou bytes de um arquivo.

Respostas aos Exercícios Guiados

1. Liste o conteúdo do diretório atual, incluindo o proprietário e as permissões, e redirecione a saída para um arquivo chamado `contents.txt` dentro de seu diretório inicial.

```
$ ls -l > contents.txt
```

2. Ordene o conteúdo do arquivo `contents.txt` a partir do diretório atual e inclua esse conteúdo ordenado no final de um novo arquivo chamado `contents-sorted.txt`

```
$ sort contents.txt >> contents-sorted.txt
```

3. Exiba as últimas 10 linhas do arquivo `/etc/passwd` e redirecione-as para um novo arquivo no diretório `Documents` do seu usuário.

```
$ tail /etc/passwd > Documents/newfile
```

4. Conte o número de palavras no arquivo `contents.txt` e inclua a saída no final de um arquivo chamado `field2.txt` em seu diretório inicial. Você precisará usar redirecionamentos de entrada e de saída.

```
$ wc < contents.txt >> field2.txt
```

5. Exiba as 5 primeiras linhas do arquivo `/etc/passwd` e classifique a saída em ordem alfabética reversa.

```
$ head -n 5 /etc/passwd | sort -r
```

6. Usando o arquivo `contents.txt` criado anteriormente, conte o número de caracteres das últimas 9 linhas.

```
$ tail -n 9 contents.txt | wc -c
531
```

7. Conte o número de arquivos chamados `test` no diretório `/usr/share` e em seus subdiretórios. Nota: cada linha da saída do comando `find` representa um arquivo.

```
$ find /usr/share -name test | wc -l
125
```

Respostas aos Exercícios Exploratórios

1. Selecione o segundo campo do arquivo `contents.txt` e redirecione a saída padrão e a saída de erro para outro arquivo chamado `field1.txt`.

   ```
   $ cut -f 2 -d " " contents.txt &> field1.txt
   ```

2. Usando o operador de redirecionamento de entrada e o comando `tr`, exclua os hífens (`-`) do arquivo `contents.txt`.

   ```
   $ tr -d "-" < contents.txt
   ```

3. Qual é a maior vantagem de redirecionar apenas os erros para um arquivo?

   O redirecionamento dos erros para um arquivo ajuda a manter um arquivo de log monitorado com frequência.

4. Substitua todos os espaços repetidos no arquivo `contents.txt` classificado em ordem alfabética por um único espaço.

   ```
   $ sort contents.txt | tr -s " "
   ```

5. Em uma linha de comando, elimine os espaços repetidos (como feito no exercício anterior), selecione o nono campo e ordene-o em ordem alfabética reversa e sem distinção entre maiúsculas e minúsculas. Quantos pipes você teve de usar?

   ```
   $ cat contents.txt | tr -s " " | cut -f 9 -d " " | sort -fr
   ```

   O exercício usa 3 pipes, um para cada filtro.

3.2 Lição 2

Certificação:	Linux Essentials
Versão:	1.6
Tópico:	3 O poder da linha de comando
Objetivo:	3.2 Como buscar e extrair dados de arquivos
Lição:	2 of 2

Introdução

Nesta lição, trataremos de ferramentas para a manipulação de texto. Elas são frequentemente usadas por administradores de sistema ou programas para monitorar ou identificar automaticamente informações recorrentes específicas.

Pesquisando dentro de arquivos com `grep`

A primeira ferramenta que discutiremos nesta lição é o comando `grep`. `grep` é a abreviação da frase “global regular expression print” (impressão de expressão regular global) e sua principal funcionalidade é buscar por um padrão especificado dentro de arquivos. O comando exibe, destacada em vermelho, a linha que contém o padrão especificado.

```
$ grep bash /etc/passwd
root:x:0:0:root:/root:/bin/bash
user:x:1001:1001:User,,,,:/home/user:/bin/bash
```

O `grep`, como a maioria dos comandos, também pode ser refinado com opções. As mais comuns

são:

`-i`

a busca não diferencia maiúsculas de minúsculas

`-r`

a busca é recursiva (pesquisa todos os arquivos dentro do diretório especificado, mais seus subdiretórios)

`-c`

a busca conta o número de correspondências

`-v`

inverte o termo de correspondência para imprimir linhas que não correspondem ao termo de pesquisa

`-E`

ativa expressões regulares estendidas (necessárias para alguns dos metacaracteres mais avançados, como `|`, `+` e ?)

O `grep` tem muitas outras opções úteis. Consulte a página do manual para saber mais sobre ele.

Expressões regulares

A segunda ferramenta é muito poderosa. É usada para descrever pedacinhos de texto dentro de arquivos, também chamados de *expressões regulares*. As expressões regulares são utilíssimas na extração de dados de arquivos de texto através da construção de padrões. São comumente usadas em scripts ou em programação com linguagens de alto nível, como Perl ou Python.

Ao trabalhar com expressões regulares, é muito importante ter em mente que *todos os caracteres contam*; o padrão é escrito com o objetivo de corresponder a uma sequência específica de caracteres, conhecida como string. A maioria dos padrões usa os símbolos ASCII normais, como letras, dígitos, sinais de pontuação ou outros símbolos, mas também pode usar caracteres Unicode para corresponder a qualquer outro tipo de texto.

A lista a seguir explica os metacaracteres das expressões regulares usadas para formar os padrões.

`.`

Corresponde a qualquer caractere único (exceto nova linha)

`[abcABC]`

Corresponde a qualquer caractere dentro dos colchetes

`[^abcABC]`

Corresponde a qualquer caractere, exceto os que estão entre colchetes

`[a-z]`

Corresponde a qualquer caractere no intervalo

`[^a-z]`

Corresponde a qualquer caractere, exceto os do intervalo

`sun|moon`

Busca qualquer uma das strings listadas

`^`

Início de uma linha

`$`

Fim de uma linha

Todas as funcionalidades das expressões regulares também podem ser implementadas através do `grep`. Você pode constatar que, no exemplo acima, a palavra não está entre aspas duplas. Para impedir que o shell interprete o próprio metacaractere, é recomendável que o padrão mais complexo esteja entre aspas duplas (" "). Para fins de prática, usaremos aspas duplas ao implementar expressões regulares. As outras aspas mantêm sua funcionalidade normal, conforme discutido nas lições anteriores.

Os exemplos a seguir enfatizam a funcionalidade das expressões regulares. Precisaremos de dados que estão dentro do arquivo, e portanto o próximo conjunto de comandos somente inclui strings diferentes no arquivo `text.txt`.

```
$ echo "aaabbb1" > text.txt
$ echo "abab2" >> text.txt
$ echo "noone2" >> text.txt
$ echo "class1" >> text.txt
$ echo "alien2" >> text.txt
$ cat text.txt
aaabbb1
abab2
noone2
```

```
class1
alien2
```

O primeiro exemplo é uma combinação de buscas no arquivo sem e com expressões regulares. Para entender bem as expressões regulares, é muito importante mostrar a diferença. O primeiro comando procura pela string exata em qualquer lugar da linha, enquanto o segundo comando procura por conjuntos de caracteres que contenham qualquer um dos caracteres entre colchetes. Portanto, os resultados dos comandos são diferentes.

```
$ grep "ab" text.txt
aaabbb1
abab2
$ grep "[ab]" text.txt
aaabbb1
abab2
class1
alien2
```

O segundo conjunto de exemplos mostra a aplicação do metacaractere de início e fim da linha. É muito importante especificar a necessidade de colocar os 2 caracteres no lugar certo dentro da expressão. Ao se especificar o início da linha, o metacaractere precisa vir antes da expressão; já ao se especificar o final da linha, o metacaractere deve ser posto após a expressão.

```
$ grep "^a" text.txt
aaabbb1
abab2
alien2
$ grep "2$" text.txt
abab2
noone2
alien2
```

Além dos metacaracteres explicados anteriormente, as expressões regulares também possuem metacaracteres que permitem a multiplicação do padrão especificado anteriormente:

`*`

Zero ou mais do padrão anterior

`+`

Um ou mais do padrão anterior

`?`

Zero ou um do padrão anterior

Para os metacaracteres multiplicadores, o comando abaixo busca uma sequência que contenha `ab`, um único caractere e um ou mais dos caracteres encontrados anteriormente. O resultado mostra que o `grep` encontrou a string `aaabbb1`, correspondendo à parte do `abbb`, bem como `abab2`. Como o caractere `+` é um caractere de expressão regular *estendida* , precisamos passar a opção `-E` para o comando `grep`.

```
$ grep -E "ab.+" text.txt
aaabbb1
abab2
```

A maioria dos metacaracteres é autoexplicativa, mas no início eles podem ser meio difíceis de assimilar. Os exemplos anteriores representam uma pequena parte da funcionalidade das expressões regulares. Experimente todos os metacaracteres da tabela acima para entender melhor como eles funcionam.

Exercícios Guiados

Usando `grep` e o arquivo `/usr/share/hunspell/en_US.dic`, encontre as linhas que correspondem aos seguintes critérios:

1. Todas as linhas que contêm a palavra `cat` em qualquer lugar da linha.

2. Todas as linhas que não contêm nenhum dos seguintes caracteres: `sawgtfixk`.

3. Todas as linhas que começam com 3 letras quaisquer e a palavra `dig`.

4. Todas as linhas que terminam com pelo menos um `e`.

5. Todas as linhas que contêm uma das seguintes palavras: `org` , `kay` ou `tuna`.

6. Número de linhas que começam com um ou nenhum `c` seguido pela string `ati`.

Exercícios Exploratórios

1. Encontre a expressão regular que corresponde às palavras na linha “Incluir” e que não corresponde às da linha “Excluir”::
   - Incluir: `pot`, `spot`, `apot`

     Excluir: `potic`, `spots`, `potatoe`
   - Incluir: `arp99`, `apple`, `zipper`

     Excluir: `zoo`, `arive`, `attack`
   - Incluir: `arcane`, `capper`, `zoology`

     Excluir: `air`, `coper`, `zoloc`
   - Incluir: `0th/pt`, `3th/tc`, `9th/pt`

     Excluir: `0/nm`, `3/nm`, `9/nm`
   - Incluir: `Hawaii`, `Dario`, `Ramiro`

     Excluir: `hawaii`, `Ian`, `Alice`
2. Que outro comando útil é comumente usado para pesquisar dentro de arquivos? Quais funcionalidades adicionais ele oferece?
3. Escolha um dos exemplos da lição anterior e tente buscar por um padrão específico na saída do comando com a ajuda do `grep`.

Resumo

Nesta lição, você aprendeu:

- Metacaracteres em expressões regulares
- Como criar padrões com expressões regulares
- Como buscar dentro de arquivos

Comandos usados nos exercícios:

`grep`
: Busca por caracteres ou sequências dentro de um arquivo

Respostas aos Exercícios Guiados

Usando `grep` e o arquivo `/usr/share/hunspell/en_US.dic`, encontre as linhas que correspondem aos seguintes critérios:

1. Todas as linhas que contêm a palavra `cat` em qualquer lugar da linha.

```
$ grep "cat" /usr/share/hunspell/en_US.dic
Alcatraz/M
Decatur/M
Hecate/M
...
```

2. Todas as linhas que não contêm nenhum dos seguintes caracteres: `sawgtfixk`.

```
$ grep -v "[sawgtfixk]" /usr/share/hunspell/en_US.dic
49269
0/nm
1/n1
2/nm
2nd/p
3/nm
3rd/p
4/nm
5/nm
6/nm
7/nm
8/nm
...
```

3. Todas as linhas que começam com 3 letras quaisquer e a palavra `dig`.

```
$ grep "^...dig" /usr/share/hunspell/en_US.dic
cardigan/SM
condign
predigest/GDS
...
```

4. Todas as linhas que terminam com pelo menos um `e`.

```
$ grep -E "e+$" /usr/share/hunspell/en_US.dic
Anglicize
Anglophobe
Anthropocene
...
```

5. Todas as linhas que contêm uma das seguintes palavras: `org` , `kay` ou `tuna`.

```
$ grep -E "org|kay|tuna" /usr/share/hunspell/en_US.dic
Borg/SM
George/MS
Tokay/M
fortunate/UY
...
```

6. Número de linhas que começam com um ou nenhum `c` seguido pela string `ati`.

```
$ grep -cE "^c?ati" /usr/share/hunspell/en_US.dic
3
```

Respostas aos Exercícios Exploratórios

1. Encontre a expressão regular que corresponde às palavras na linha “Incluir” e que não corresponde às da linha “Excluir”:
   - Incluir: `pot`, `spot`, `apot`

     Excluir: `potic`, `spots`, `potatoe`

     Resposta: `pot$`
   - Incluir: `arp99`, `apple`, `zipper`

     Excluir: `zoo`, `arive`, `attack`

     Resposta: `p+`
   - Incluir: `arcane`, `capper`, `zoology`

     Excluir: `air`, `coper`, `zoloc`

     Resposta: `arc|cap|zoo`
   - Incluir: `0th/pt`, `3th/tc`, `9th/pt`

     Excluir: `0/nm`, `3/nm`, `9/nm`

     Resposta: `[0-9]th.+`
   - Incluir: `Hawaii`, `Dario`, `Ramiro`

     Excluir: `hawaii`, `Ian`, `Alice`

     Resposta: `^[A-Z]a.*i+`
2. Que outro comando útil é comumente usado para pesquisar dentro de arquivos? Quais funcionalidades adicionais ele oferece?

   O comando `sed`. Ele é capaz de encontrar e substituir caracteres ou conjuntos de caracteres dentro de um arquivo.
3. Escolha um dos exemplos da lição anterior e tente buscar por um padrão específico na saída do comando com a ajuda do `grep`.

   Peguei uma das respostas dos Exercícios Exploratórios e busquei pela linha que tem as

permissões de grupo para leitura, escrita e execução. Sua resposta pode ser diferente, dependendo do comando que você escolher e do padrão que criar.

```
$ cat contents.txt | tr -s " " | grep "^....rwx"
```

Este exercício serve para demonstrar que o `grep` também pode receber entrada de diferentes comandos e ajudar a filtrar as informações geradas.

3.3 Transformando comandos em Scripts

Referência ao LPI objectivo

Linux Essentials version 1.6, Exam 010, Objective 3.3

Peso

4

Áreas chave de conhecimento

- Shell scripting básico
- Reconhecimento de editores de texto comuns (vi e nano).

Partial list of the used files, terms and utilities

- `#!` (shebang)
- `/bin/bash`
- Variáveis
- Argumentos
- `for` loops
- `echo`
- Exit status

3.3 Lesson 1

Certificação:	Linux Essentials
Versão:	1.6
Tópico:	3 O poder da linha de comando
Objetivo:	3.3 Como transformar comandos em script
Lição:	1 de 2

Introdução

Até agora, aprendemos a executar comandos a partir do shell, mas também é possível inserir comandos em um arquivo e depois configurá-lo para ser executável. Quando executamos o arquivo, os comandos são rodados um após o outro. Esses arquivos executáveis são chamados *scripts* e representam uma ferramenta absolutamente essencial para qualquer administrador de sistema Linux. Basicamente, podemos dizer que o Bash, além de um shell, é uma verdadeira linguagem de programação.

Exibindo a saída

Vamos começar demonstrando um comando que você deve ter visto nas lições anteriores: o `echo` imprime na tela um argumento na saída padrão.

```
$ echo "Hello World!"
Hello World!
```

Agora, usaremos um redirecionamento de arquivo para enviar este comando para um novo

arquivo chamado `new_script`.

```
$ echo 'echo "Hello World!"' > new_script
$ cat new_script
echo "Hello World!"
```

O arquivo `new_script` agora contém o mesmo comando de antes.

Tornando um script executável

Vamos demonstrar algumas das etapas necessárias para executar esse arquivo da maneira esperada. A primeira ideia de um usuário pode ser simplesmente digitar o nome do script, assim como digitaria o nome de qualquer outro comando:

```
$ new_script
/bin/bash: new_script: command not found
```

Sabemos que `new_script` existe em nossa localização atual, mas note que a mensagem de erro não está dizendo que o *arquivo* não existe, mas sim que o *comando* não existe. Seria útil discutir como o Linux lida com comandos e executáveis.

Comandos e `PATH`

Quando digitamos o comando `ls` no shell, por exemplo, estamos na verdade executando um arquivo chamado `ls` que existe em nosso sistema de arquivos. Para comprovar, use `which`:

```
$ which ls
/bin/ls
```

Seria cansativo digitar o caminho absoluto de `ls` toda vez que desejássemos examinar o conteúdo de um diretório, por isso o Bash tem uma *variável de ambiente* que contém todos os diretórios em que ficam os comandos que queremos executar. Para ver o conteúdo dessa variável, use `echo`.

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/sn
ap/bin
```

O shell espera encontrar um comando em cada um desses locais, delimitado por dois pontos (`:`).

Note que `/bin` está presente, mas é seguro supor que nossa localização atual não está. O shell procurará por `new_script` em cada um desses diretórios, mas não o encontrará e, portanto, exibirá a mensagem de erro que vimos acima.

Existem três soluções para esse problema: podemos mover `new_script` para um dos diretórios de `PATH`, podemos adicionar o diretório atual a `PATH`, ou podemos mudar a maneira de chamar o script. A última solução é a mais fácil, pois simplesmente requer que especifiquemos o *local atual* ao chamar o script usando ponto e barra (`./`).

```
$ ./new_script
/bin/bash: ./new_script: Permission denied
```

A mensagem de erro já está diferente, o que indica que fizemos alguns progressos.

Permissões de execução

A primeira investigação que um usuário deve fazer nesse caso é usar `ls -l` para examinar o arquivo:

```
$ ls -l new_script
-rw-rw-r-- 1 user user 20 Apr 30 12:12 new_script
```

Podemos ver que as permissões para este arquivo estão definidas como `664` por padrão. Ainda não configuramos este arquivo para ter *permissões de execução*.

```
$ chmod +x new_script
$ ls -l new_script
-rwxrwxr-x 1 user user 20 Apr 30 12:12 new_script
```

Este comando concede permissões de execução a todos os usuários. Esteja ciente de que isso pode ser um risco de segurança, mas, por enquanto, esse é um nível aceitável de permissão.

```
$ ./new_script
Hello World!
```

Agora já é possível executar nosso script.

Definindo o intérprete

Como demonstramos, é possível simplesmente inserir texto em um arquivo, configurá-lo como executável e executá-lo. O `new_script` ainda é tecnicamente um arquivo de texto normal, mas pode ser interpretado pelo Bash. Mas e se ele estiver escrito em Perl ou Python?

É extremamente recomendável especificar o tipo de intérprete que queremos usar na primeira linha de um script. Essa linha é chamada de *bang line*, ou, mais comumente, *shebang*. Ela indica ao sistema como queremos que aquele arquivo seja executado. Como estamos aprendendo Bash, usaremos o caminho absoluto para o nosso executável do Bash, mais uma vez usando `which`:

```
$ which bash
/bin/bash
```

Nosso shebang começa com um sinal de hash e um ponto de exclamação, seguidos pelo caminho absoluto acima. Vamos abrir `new_script` em um editor de texto e inserir o shebang. Vamos também aproveitar a oportunidade para inserir um *comentário* em nosso script. Os comentários são ignorados pelo intérprete. Eles são escritos para o benefício de outros usuários que queiram entender seu script.

```
#!/bin/bash

# Este é nosso primeiro comentário. Também é recomendável documentar todos os scripts.

echo "Hello World!"
```

Também faremos uma alteração adicional no nome do arquivo: salvaremos o arquivo como `new_script.sh`. O sufixo do arquivo ".sh" não altera em nada a execução do arquivo. É uma convenção que os scripts do bash sejam rotulados com `.sh` ou `.bash` para identificá-los mais facilmente, assim como os scripts em Python são geralmente identificados com o sufixo `.py`.

Editores de texto comuns

Os usuários do Linux geralmente precisam trabalhar em ambientes nos quais editores de texto gráficos não estão disponíveis. Portanto, é altamente recomendável aprender ao menos o básico sobre a edição de arquivos de texto na linha de comando. Dois dos editores de texto mais comuns são `vi` e `nano`.

vi

O `vi` é um editor de texto venerável, que vem instalado por padrão em quase todos os sistemas Linux existentes. O `vi` gerou um clone chamado *vi IMproved* ou `vim`, que tem algumas funcionalidades adicionais, mas mantém a interface do `vi`. Embora trabalhar com o `vi` seja assustador para um novo usuário, o editor é popular e amado pelos usuários que conhecem seus muitos recursos.

A diferença mais importante entre o `vi` e aplicativos como o Bloco de Notas é que o `vi` possui três modos diferentes. Na inicialização, as teclas `H`, `J`, `K` e `L` são usadas para navegar, e não para digitar. Nesse _ modo de navegação_, pressionamos `I` para entrar no *modo de inserção*. A partir daí, você pode digitar normalmente. Para sair do modo de inserção, pressione `Esc` para retornar ao *modo de navegação*. No *modo de navegação*, pressione `:` para entrar no *modo de comando*. Nesse modo, podemos salvar, excluir, sair ou alterar opções.

Embora o `vi` tenha uma curva de aprendizado, os diferentes modos permitem que um usuário experiente seja mais eficiente do que com outros editores.

nano

O `nano` é uma ferramenta mais recente, criada para ser mais simples e fácil de usar do que o `vi`. O `nano` não oferece modos diferentes. Em vez disso, um usuário pode começar a digitar já na inicialização, usando `Ctrl` para acessar as ferramentas impressas na parte inferior da tela.

```
                        [ Welcome to nano.  For basic help, type Ctrl+G. ]
^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos    M-U Undo
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line M-E Redo
```

Os editores de texto são uma questão de preferência pessoal, e o editor que você escolher não fará diferença para esta lição. Mas vale muito a pena procurar conhecer e se familiarizar com um ou mais editores de texto.

Variáveis

As variáveis são uma parte importante de qualquer linguagem de programação, e o mesmo vale para o Bash. Quando iniciamos uma nova sessão no terminal, o shell já define algumas variáveis automaticamente. A variável `PATH` é um bom exemplo. Nós as chamamos de *variáveis de ambiente*, porque geralmente definem características do nosso ambiente shell. É possível modificar e adicionar variáveis de ambiente, mas por enquanto vamos nos concentrar na configuração de variáveis dentro do nosso script.

Vamos modificar o script da seguinte forma:

```
#!/bin/bash

# Este é nosso primeiro comentário. Também é recomendável documentar todos os scripts.

username=Carol

echo "Hello $username!"
```

Neste caso, criamos uma *variável* chamada `username` e atribuímos a ela o *valor* `Carol`. Note que não há espaços entre o nome da variável, o sinal de igual ou o valor atribuído.

Na linha seguinte, usamos o comando `echo` com a variável, mas há um cifrão (`$`) na frente do nome da variável. Isso é importante, pois indica para o shell que queremos tratar `username` como uma variável, e não apenas como uma palavra normal. Ao digitar `$username` no comando, indicamos que queremos executar uma *substituição*, trocando o *nome* de uma variável pelo *valor* atribuído a essa variável.

Ao executar o novo script, obtemos esta saída:

```
$ ./new_script.sh
Hello Carol!
```

- As variáveis devem conter apenas caracteres alfanuméricos ou sublinhados (underline) e diferenciam maiúsculas de minúsculas. `Username` e `username` serão tratados como variáveis distintas.
- A substituição de variáveis também pode ter o formato `${username}`, com a adição de `{ }`. Isso também é aceitável.
- As variáveis no Bash têm um *tipo implícito*, sendo consideradas strings. Isso significa que executar funções matemáticas no Bash é mais complicado do que seria em outras linguagens de programação, como C/C++:

```
#!/bin/bash

# Este é nosso primeiro comentário. Também é recomendável documentar todos os scripts.

username=Carol
x=2
y=4
```

```
z=$x+$y
echo "Hello $username!"
echo "$x + $y"
echo "$z"
```

```
$ ./new_script.sh
Hello Carol!
2 + 4
2+4
```

Usando aspas com variáveis

Vamos fazer a seguinte alteração no valor da nossa variável `username`:

```
#!/bin/bash

# Este é nosso primeiro comentário. Também é recomendável documentar todos os scripts.

username=Carol Smith

echo "Hello $username!"
```

A execução desse script resultará em um erro:

```
$ ./new_script.sh
./new_script.sh: line 5: Smith: command not found
Hello !
```

Lembre-se de que o Bash é um intérprete e, como tal, *interpreta* o script linha por linha. Neste caso, ele interpretou corretamente `username=Carol` como a definição de uma variável `username` com o valor `Carol`. Mas, em seguida, ele interpretou que o espaço indicava o final dessa tarefa e que `Smith` era o nome de um comando. Para que o espaço e o nome `Smith` sejam incluídos como o novo valor da variável, colocamos aspas duplas (`"`) ao redor do nome.

```
#!/bin/bash

# Este é nosso primeiro comentário. Também é recomendável documentar todos os scripts.

username="Carol Smith"
```

```
echo "Hello $username!"
```

```
$ ./new_script.sh
Hello Carol Smith!
```

Uma coisa importante a ser observada no Bash é que aspas duplas e aspas simples (`'`) se comportam de maneira muito diferente. As aspas duplas são consideradas “fracas” porque permitem que o intérprete efetue a substituição dentro das aspas. Aspas simples são consideradas “fortes” por impedirem a substituição. Considere o seguinte exemplo:

```
#!/bin/bash

# Este é nosso primeiro comentário. Também é recomendável documentar todos os scripts.

username="Carol Smith"

echo "Hello $username!"
echo 'Hello $username!'
```

```
$ ./new_script.sh
Hello Carol Smith!
Hello $username!
```

No segundo comando `echo`, o intérprete foi impedido de substituir `$username` por `Carol Smith`, e assim a saída é interpretada literalmente.

Argumentos

Você já aprendeu a usar argumentos nos utilitários principais do Linux. Por exemplo, `rm testfile` contém tanto o executável `rm` quanto um argumento `testfile`. Os argumentos podem ser passados para o script na execução e modificam o comportamento do script. É fácil implementá-los.

```
#!/bin/bash

# Este é nosso primeiro comentário. Também é recomendável documentar todos os scripts.

username=$1
```

```
echo "Hello $username!"
```

Em vez de atribuir um valor a `username` diretamente dentro do script, atribuímos a ele o valor de uma nova variável `$1`. Ela faz referência ao valor do *primeiro argumento*.

```
$ ./new_script.sh Carol
Hello Carol!
```

Os nove primeiros argumentos são tratados dessa maneira. Há modos de lidar com mais de nove argumentos, mas isso está fora do alcance desta lição. Eis um exemplo usando apenas dois argumentos:

```
#!/bin/bash

# Este é nosso primeiro comentário. Também é recomendável documentar todos os scripts.

username1=$1
username2=$2
echo "Hello $username1 and $username2!"
```

```
$ ./new_script.sh Carol Dave
Hello Carol and Dave!
```

Existe uma consideração importante ao usar argumentos: no exemplo acima, temos dois argumentos, `Carol` e `Dave`, atribuídos a `$1` e `$2`, respectivamente. Se o segundo argumento estiver ausente, por exemplo, o shell não emitirá uma mensagem de erro. O valor de `$2` será simplesmente *nulo*, ou nada.

```
$ ./new_script.sh Carol
Hello Carol and !
```

Em nosso caso, seria recomendável introduzir alguma lógica em nosso script para que diferentes *condições* afetem a *saída* que desejamos imprimir. Começaremos introduzindo outra variável útil e, em seguida, criaremos *declarações if*.

Retorno do número de argumentos

Enquanto variáveis como `$1` e `$2` contêm o valor dos argumentos posicionais, a variável `$#` contém o *número de argumentos*.

```
#!/bin/bash

# Este é nosso primeiro comentário. Também é recomendável documentar todos os scripts.

username=$1

echo "Hello $username!"
echo "Number of arguments: $#."
```

```
$ ./new_script.sh Carol Dave
Hello Carol!
Number of arguments: 2.
```

Lógica Condicional

O uso da lógica condicional na programação é um tópico vasto que não será abordado em profundidade nesta lição. Vamos nos concentrar na *sintaxe* dos condicionais no Bash, que é diferente da maioria das outras linguagens de programação.

Primeiro, vamos rever nosso objetivo. Temos um script simples capaz de imprimir uma saudação para um único usuário. Se houver algo diferente de um único usuário, temos de imprimir uma mensagem de erro.

- A *condição* que estamos testando é o número de usuários, que está contido na variável `$#`. Gostaríamos de saber se o valor de `$#` é `1`.
- Se a condição for *verdadeira*, a *ação* executada será saudar o usuário.
- Se a condição for *falsa*, imprimiremos uma mensagem de erro.

Agora que a lógica está clara, vamos nos concentrar na *sintaxe* necessária para implementá-la.

```
#!/bin/bash

# Um script simples para saudar um único usuário.
```

```
if [ $# -eq 1 ]
then
  username=$1

  echo "Hello $username!"
else
  echo "Please enter only one argument."
fi
echo "Number of arguments: $#."
```

A lógica condicional está contida entre `if` e `fi`. A condição a ser testada fica entre colchetes `[ ]` e a ação a tomar caso a condição seja verdadeira é indicada após `then`. Observe os espaços entre os colchetes e a lógica contida neles. A omissão desses espaços causará erros.

Esse script exibirá a nossa saudação *ou* a mensagem de erro. Mas ele sempre imprimirá a linha `Number of arguments`.

```
$ ./new_script.sh
Please enter only one argument.
Number of arguments: 0.
$ ./new_script.sh Carol
Hello Carol!
Number of arguments: 1.
```

Tome nota da declaração `if`. Usamos `-eq` para fazer uma *comparação numérica*. Nesse caso, estamos testando se o valor de `$#` é *igual* a um. As outras comparações que podemos realizar são:

`-ne`
: Diferente de

`-gt`
: Maior que

`-ge`
: Maior que ou igual a

`-lt`
: Menos que

`-le`
: Menos que ou igual a

Exercícios guiados

1. Um usuário insere o seguinte no shell:

```
$ PATH=~/scripts
$ ls
Command 'ls' is available in '/bin/ls'
The command could not be located because '/bin' is not included in the PATH environment
variable.
ls: command not found
```

   - O que o usuário fez?

   - Qual comando pode combinar o valor atual de `PATH` com o novo diretório `~/scripts`?

2. Considere o script a seguir. Note que ele usa `elif` para conferir uma segunda condição:

```
>  /!bin/bash

> fruit1 = Apples
> fruit2 = Oranges

  if [ $1 -lt $# ]
  then
    echo "This is like comparing $fruit1 and $fruit2!"
> elif [$1 -gt $2 ]
  then
>   echo '$fruit1 win!'
  else
>   echo "Fruit2 win!"
> done
```

   - As linhas marcadas com um `>` contêm erros. Corrija esses erros.

3. Qual será a saída nas seguintes situações?

```
$ ./guided1.sh 3 0
```

```
$ ./guided1.sh 2 4
```

```
$ ./guided1.sh 0 1
```

Exercícios Exploratórios

1. Escreva um script simples que verifique se exatamente dois argumentos são passados. Se for o caso, imprima os argumentos em ordem inversa. Considere o exemplo (nota: seu código pode ser diferente do nosso, mas deve produzir uma saída idêntica):

```
if [ $1 == $number ]
then
  echo "True!"
fi
```

2. Este código está correto, mas não é uma comparação numérica. Pesquise na internet para descobrir como este código é diferente de usar `-eq`.

3. Existe uma variável de ambiente para imprimir o diretório atual. Use `env` para descobrir o nome dessa variável.

4. Usando o que aprendeu nas questões 2 e 3, escreva um script curto que aceite um argumento. Se um argumento for passado, confira se ele corresponde ao nome do diretório atual. Se a resposta for sim, imprima `yes`. Caso contrário, imprima `no`.

Resumo

Nesta lição, você aprendeu:

- Como criar e executar scripts simples
- Como usar um shebang para especificar um intérprete
- Como definir e usar variáveis dentro de scripts
- Como lidar com argumentos em scripts
- Como construir declarações `if`
- Como comparar números usando operadores numéricos

Comandos usados nos exercícios:

`echo`
: Imprime uma string na saída padrão.

`env`
: Imprime todas as variáveis de ambiente na saída padrão.

`which`
: Imprime o caminho absoluto de um comando.

`chmod`
: Altera as permissões de um arquivo.

Variáveis especiais usadas nos exercícios:

`$1, $2, … $9`
: Contêm argumentos posicionais passados para o script.

`$#`
: Contém o número de argumentos passados para o script.

`$PATH`
: Contém os diretórios que possuem executáveis usados pelo sistema.

Operadores usados nos exercícios:

`-ne`

Diferente de

`-gt`

Maior que

`-ge`

Maior que ou igual a

`-lt`

Menos que

`-le`

Menos que ou igual a

Respostas aos Exercícios Guiados

1. Um usuário insere o seguinte no shell:

```
$ PATH=~/scripts
$ ls
Command 'ls' is available in '/bin/ls'
The command could not be located because '/bin' is not included in the PATH environment
variable.
ls: command not found
```

   - O que o usuário fez?

     O usuário sobrescreveu o conteúdo de PATH com o diretório `~/scripts`. O comando `ls` não pode mais ser encontrado, já que não está contido em PATH. Note que essa alteração afeta somente a sessão atual; basta se deslogar e se relogar novamente para reverter a mudança.

   - Qual comando pode combinar o valor atual de `PATH` com o novo diretório `~/scripts`?

     `PATH=$PATH:~/scripts`

2. Considere o script a seguir. Note que ele usa `elif` para conferir uma segunda condição:

```
>  /!bin/bash

> fruit1 = Apples
> fruit2 = Oranges

  if [ $1 -lt $# ]
  then
    echo "This is like comparing $fruit1 and $fruit2!"
> elif [$1 -gt $2 ]
  then
>   echo '$fruit1 win!'
  else
>   echo "Fruit2 win!"
> done
```

   - As linhas marcadas com um `>` contêm erros. Corrija esses erros.

```
#!/bin/bash
```

```
fruit1=Apples
fruit2=Oranges

if [ $1 -lt $# ]
then
  echo "This is like comparing $fruit1 and $fruit2!"
elif [ $1 -gt $2 ]
then
  echo "$fruit1 win!"
else
  echo "$fruit2 win!"
fi
```

3. Qual será a saída nas seguintes situações?

```
$ ./guided1.sh 3 0
```

`Apples win!`

```
$ ./guided1.sh 2 4
```

`Oranges win!`

```
$ ./guided1.sh 0 1
```

`This is like comparing Apples and Oranges!`

Respostas aos Exercícios Exploratórios

1. Escreva um script simples que verifique se exatamente dois argumentos são passados. Se for o caso, imprima os argumentos em ordem inversa. Considere o exemplo (nota: seu código pode ser diferente do nosso, mas deve produzir uma saída idêntica):

```
if [ $1 == $number ]
then
  echo "True!"
fi
```

```
#!/bin/bash

if [ $# -ne 2 ]
then
  echo "Error"
else
  echo "$2 $1"
fi
```

2. Este código está correto, mas não é uma comparação numérica. Pesquise na internet para descobrir como este código é diferente de usar `-eq`.

   Podemos comparar *strings* com `==`. Ou seja, se os caracteres de ambas as variáveis corresponderem exatamente, a condição é verdadeira.

`abc == abc`	*true*
`abc == ABC`	*false*
`1 == 1`	*true*
`1+1 == 2`	*false*

   A comparação de strings provoca comportamentos inesperados se estivermos testando com números.

3. Existe uma variável de ambiente para imprimir o diretório atual. Use `env` para descobrir o nome dessa variável.

   `PWD`

4. Usando o que aprendeu nas questões 2 e 3, escreva um script curto que aceite um argumento. Se um argumento for passado, confira se ele corresponde ao nome do diretório atual. Se a resposta for sim, imprima `yes`. Caso contrário, imprima `no`.

```
#!/bin/bash

if [ "$1" == "$PWD" ]
then
  echo "yes"
else
  echo "no"
fi
```

3.3 Lição 2

Certificação:	Linux Essentials
Versão:	1.6
Tópico:	3 O poder da linha de comando
Objetivo:	3.3 Como transformar comandos em script
Lição:	2 of 2

Introdução

Na última seção, usamos este exemplo simples para demonstrar a criação de scripts no Bash:

```
#!/bin/bash

# Um script simples para saudar um único usuário.

if [ $# -eq 1 ]
then
  username=$1

  echo "Hello $username!"
else
  echo "Please enter only one argument."
fi
echo "Number of arguments: $#."
```

- Todos os scripts devem começar com um *shebang*, que define o caminho para o intérprete.

- Todos os scripts devem incluir comentários para descrever seu uso.
- Este script em particular trabalha com um *argumento*, que é passado para o script quando chamado.
- Este script contém uma *declaração if*, que testa as condições de uma variável interna `$#`. Essa variável foi configurada para o número de argumentos.
- Se o número de argumentos passados para o script for igual a 1, o valor do primeiro argumento é passado para uma nova variável chamada `username` e o script ecoa uma saudação ao usuário. Caso contrário, uma mensagem de erro é exibida.
- Finalmente, o script ecoa o número de argumentos. Isso é útil para a depuração (debugging).

Este é um exemplo útil para começar a explicar alguns dos outros recursos dos scripts Bash.

Códigos de saída

Você deve ter notado que nosso script tem dois estados possíveis: ou ele imprime `"Hello <usuário>!"` ou uma mensagem de erro. Isso é bastante normal para muitos dos utilitários fundamentais. Veja o `cat`, que você já deve estar conhecendo bem.

Vamos comparar um uso bem-sucedido do `cat` com uma situação em que ele falha. Relembrando, o exemplo acima é um script chamado `new_script.sh`.

```
$ cat -n new_script.sh

     1  #!/bin/bash
     2
     3  # Um script simples para saudar um único usuário.
     4
     5  if [ $# -eq 1 ]
     6  then
     7    username=$1
     8
     9    echo "Hello $username!"
    10  else
    11    echo "Please enter only one argument."
    12  fi
    13  echo "Number of arguments: $#."
```

Este comando é bem-sucedido; vemos que a flag `-n` também imprimiu os números de linha. Isso é muito útil na depuração de scripts, mas note eles não fazem parte do script.

Agora vamos verificar o valor de uma nova variável interna `$?`. Por enquanto, observe somente a saída:

```
$ echo $?
0
```

Agora, vamos considerar uma situação em que o `cat` não funcionará. Primeiro, veremos uma mensagem de erro e, em seguida, verificaremos o valor de `$?`.

```
$ cat -n dummyfile.sh
cat: dummyfile.sh: No such file or directory
$ echo $?
1
```

A explicação para esse comportamento é a seguinte: qualquer execução do utilitário `cat` retornará um *código de saída*. O código de saída informa se o comando foi bem-sucedido ou se ocorreu um erro. Um código de saída *zero* indica que o comando foi concluído com êxito. Isso vale para quase todos os comandos do Linux com os quais trabalhamos. Qualquer outro código de saída indica algum tipo de erro. O código de saída do *último comando a ser executado* será armazenado na variável `$?`.

Os códigos de saída geralmente não são vistos por usuários humanos, mas são muito úteis ao escrever scripts. Considere um script que serve para copiar arquivos para uma unidade de rede remota. A tarefa de cópia pode falhar de diversas maneiras: por exemplo, a máquina local talvez não esteja conectada à rede ou a unidade remota pode estar cheia. Graças ao código de saída do nosso utilitário de cópia, podemos alertar o usuário sobre esses problemas ao executar o script.

É sempre uma boa ideia implementar códigos de saída, e é o que faremos agora. Temos dois caminhos em nosso script, um sucesso e um fracasso. Vamos usar zero para indicar o sucesso e um para indicar o fracasso.

```
1  #!/bin/bash
2
3  # Um script simples para saudar um único usuário.
4
5  if [ $# -eq 1 ]
6  then
7    username=$1
8
9    echo "Hello $username!"
```

```
10    exit 0
11  else
12    echo "Please enter only one argument."
13    exit 1
14  fi
15  echo "Number of arguments: $#."
```

```
$ ./new_script.sh Carol
Hello Carol!
$ echo $?
0
```

Note que o comando `echo` na linha 15 foi totalmente ignorado. O uso de `exit` encerra o script imediatamente e, portanto, essa linha nunca é encontrada.

Como manipular múltiplos argumentos

Até agora, nosso script pode tratar apenas de um único nome de usuário por vez. Qualquer número de argumentos além de um causará um erro. Vamos explorar como é possível tornar esse script mais versátil.

O primeiro instinto de um usuário pode ser usar mais variáveis posicionais como `$2`, `$3` e assim por diante. Infelizmente, não podemos prever o número de argumentos que um usuário pode querer usar. Para resolver esse problema, será útil introduzir mais variáveis internas.

Vamos modificar a lógica do nosso script. Se houver zero argumentos, aparecerá um erro, mas qualquer outro número de argumentos fará o script ser bem-sucedido. Este novo script será chamado `friendly2.sh`.

```
 1  #!/bin/bash
 2
 3  # um script simples para saudar os usuários
 4
 5  if [ $# -eq 0 ]
 6  then
 7    echo "Please enter at least one user to greet."
 8    exit 1
 9  else
10    echo "Hello $@!"
11    exit 0
```

```
    12  fi
```

```
$ ./friendly2.sh Carol Dave Henry
Hello Carol Dave Henry!
```

Existem duas variáveis internas que contêm todos os argumentos passados para o script: `$@` e `$*`. Na maioria das vezes, ambas se comportam da mesma forma. O Bash analisará (*parse*) os argumentos e separará cada argumento quando encontrar um espaço entre eles. De fato, o conteúdo de `$@` é o seguinte:

`0`	`1`	`2`
`Carol`	`Dave`	`Henry`

Se você conhece outras linguagens de programação, vai reconhecer esse tipo de variável como um *arranjo* (array). Para criar arranjos no Bash, basta pôr um espaço entre elementos como a variável `FILES` no script `arraytest` abaixo:

```
FILES="/usr/sbin/accept /usr/sbin/pwck/ usr/sbin/chroot"
```

Ele contém uma lista de diversos itens. Por enquanto, ele não é muito útil, porque ainda não introduzimos nenhuma maneira de gerir esses itens individualmente.

Loops for

Vamos voltar ao exemplo `arraytest`, mostrado anteriormente. Se você se lembra, neste exemplo estamos especificando um arranjo chamado FILES. O que precisamos é de uma maneira de "desdobrar" esta variável e acessar cada valor individual, um após o outro. Para isso, usaremos uma estrutura chamada *loop for*, presente em todas as linguagens de programação. Existem duas variáveis às quais nos referiremos: uma é o intervalo e a outra é o valor individual em que estamos trabalhando no momento. Este é o script em sua totalidade:

```
#!/bin/bash

FILES="/usr/sbin/accept /usr/sbin/pwck/ usr/sbin/chroot"

for file in $FILES
do
  ls -lh $file
```

```
done
```

```
$ ./arraytest
lrwxrwxrwx 1 root root 10 Apr 24 11:02 /usr/sbin/accept -> cupsaccept
-rwxr-xr-x 1 root root 54K Mar 22 14:32 /usr/sbin/pwck
-rwxr-xr-x 1 root root 43K Jan 14 07:17 /usr/sbin/chroot
```

Se você consultar novamente o exemplo `friendly2.sh` acima, verá que estamos trabalhando com um intervalo de valores contidos dentro de uma única variável `$@`. Por uma questão de clareza, chamaremos a última variável de `username`. Nosso script fica assim:

```
 1  #!/bin/bash
 2
 3  # um script simples para saudar os usuários
 4
 5  if [ $# -eq 0 ]
 6  then
 7    echo "Please enter at least one user to greet."
 8    exit 1
 9  else
10    for username in $@
11    do
12      echo "Hello $username!"
13    done
14    exit 0
15  fi
```

Lembre-se de que a variável definida aqui pode ter qualquer nome, e que todas as linhas entre `do`... `done` serão executadas uma vez para cada elemento do arranjo. Vamos observar a saída de nosso script:

```
$ ./friendly2.sh Carol Dave Henry
Hello Carol!
Hello Dave!
Hello Henry!
```

Agora vamos supor que queremos fazer a saída parecer um pouco mais humana. Queremos que a saudação apareça em uma só linha.

```
 1  #!/bin/bash
```

```
 2
 3  # um script simples para saudar os usuários
 4
 5  if [ $# -eq 0 ]
 6  then
 7    echo "Please enter at least one user to greet."
 8    exit 1
 9  else
10    echo -n "Hello $1"
11    shift
12    for username in $@
13    do
14      echo -n ", and $username"
15    done
16    echo "!"
17    exit 0
18  fi
```

A notar:

- Ao usar `-n` com `echo` , *suprimimos a nova linha* após a impressão. Isso significa que todos os ecos serão impressos na mesma linha, e a nova linha será impressa somente após o `!` na linha 16.
- O comando `shift` removerá o primeiro elemento do arranjo, de modo que:

`0`	`1`	`2`
`Carol`	`Dave`	`Henry`

Se torna:

`0`	`1`
`Dave`	`Henry`

Observemos a saída:

```
$ ./friendly2.sh Carol
Hello Carol!
$ ./friendly2.sh Carol Dave Henry
Hello Carol, and Dave, and Henry!
```

Verificando erros com expressões regulares

Podemos querer verificar todos os argumentos que o usuário está inserindo. Por exemplo, se quisermos garantir que todos os nomes passados para `friendly2.sh` contenham *apenas letras* e que quaisquer caracteres ou números especiais causem um erro. Para executar essa verificação de erro, usaremos `grep`.

Lembre-se de que podemos usar expressões regulares com `grep`.

```
$ echo Animal | grep "^[A-Za-z]*$"
Animal
$ echo $?
0
```

```
$ echo 4n1ml | grep "^[A-Za-z]*$"
$ echo $?
1
```

O `^` e o `$` indicam, respectivamente, o início e o fim da linha. O `[A-Za-z]` indica um intervalo de letras, maiúsculas ou minúsculas. O `*` é um *quantificador* e modifica o intervalo de letras para que ele vá de zero a muitas letras. Em resumo, o `grep` será bem-sucedido se a entrada consistir *somente* de letras, mas se não for o caso ele falhará.

A próxima coisa a observar é que o `grep` está retornando códigos de saída com base nas correspondências que encontra. Uma correspondência positiva retorna `0` e uma negativa retorna `1`. Podemos usar isso para testar os argumentos dentro de nosso script.

```
 1  #!/bin/bash
 2
 3  # um script simples para saudar os usuários
 4
 5  if [ $# -eq 0 ]
 6  then
 7    echo "Please enter at least one user to greet."
 8    exit 1
 9  else
10    for username in $@
11    do
12      echo $username | grep "^[A-Za-z]*$" > /dev/null
13      if [ $? -eq 1 ]
14      then
```

```
15        echo "ERROR: Names must only contains letters."
16        exit 2
17      else
18        echo "Hello $username!"
19      fi
20    done
21    exit 0
22  fi
```

Na linha 12, estamos redirecionando a saída padrão para `/dev/null`, que é uma maneira simples de suprimi-la. Não queremos ver nenhuma saída do comando `grep`, mas somente testar seu código de saída, o que acontece na linha 13. Observe também que estamos usando um código de saída `2` para indicar um argumento inválido. Geralmente, é recomendável usar códigos de saída distintos para indicar erros diferentes; assim, um usuário mais experiente pode usar esses códigos de saída para solucionar problemas.

```
$ ./friendly2.sh Carol Dave Henry
Hello Carol!
Hello Dave!
Hello Henry!
$ ./friendly2.sh 42 Carol Dave Henry
ERROR: Names must only contains letters.
$ echo $?
2
```

Exercícios guiados

1. Leia o conteúdo de `script1.sh`, abaixo:

```
#!/bin/bash

if [ $# -lt 1 ]
then
  echo "This script requires at least 1 argument."
  exit 1
fi

echo $1 | grep "^[A-Z]*$" > /dev/null
if [ $? -ne 0 ]
then
  echo "no cake for you!"
  exit 2
fi

echo "here's your cake!"
exit 0
```

   Qual a saída destes comandos?

   - `./script1.sh`
   - `echo $?`
   - `./script1.sh cake`
   - `echo $?`
   - `./script1.sh CAKE`
   - `echo $?`

2. Leia o conteúdo do arquivo `script2.sh`:

```
for filename in $1/*.txt
do
   cp $filename $filename.bak
done
```

Descreva a finalidade deste script em seu entendimento.

Exercícios Exploratórios

1. Crie um script capaz de receber qualquer número de argumentos do usuário e que imprima apenas os argumentos com números maiores que 10.

Resumo

Nesta lição, você aprendeu:

- O que são códigos de saída, o que significam e como implementá-los
- Como verificar o código de saída de um comando
- O que são os loops `for` e como usá-los com arranjos
- Como usar `grep`, expressões regulares e códigos de saída para verificar dados inseridos pelo usuário em scripts.

Comandos usados nos exercícios:

`shift`
: Remove o primeiro elemento de um arranjo.

Variáveis Especiais:

`$?`
: Contém o código de saída do último comando executado

`$@`, **`$*`**
: Contém todos os argumentos passados para o script, como um arranjo.

Respostas aos Exercícios Guiados

1. Leia o conteúdo de `script1.sh`, abaixo:

```
#!/bin/bash

if [ $# -lt 1 ]
then
  echo "This script requires at least 1 argument."
  exit 1
fi

echo $1 | grep "^[A-Z]*$" > /dev/null
if [ $? -ne 0 ]
then
  echo "no cake for you!"
  exit 2
fi

echo "here's your cake!"
exit 0
```

   Qual a saída destes comandos?

   - Comando: `./script1.sh`

     Saída: `This script requires at least 1 argument.`

   - Comando: `echo $?`

     Saída: `1`

   - Comando: `./script1.sh cake`

     Saída: `no cake for you!`

   - Comando: `echo $?`

     Saída: `2`

   - Comando: `./script1.sh CAKE`

     Saída: `here's your cake!`

- Comando: `echo $?`

  Saída: `0`

2. Leia o conteúdo do arquivo `script2.sh`:

```
for filename in $1/*.txt
do
   cp $filename $filename.bak
done
```

   Descreva a finalidade deste script em seu entendimento.

   Este script faz cópias de segurança de todos os arquivos que terminam com `.txt` em um subdiretório definido no primeiro argumento.

Respostas aos Exercícios Exploratórios

1. Crie um script capaz de receber qualquer número de argumentos do usuário e que imprima apenas os argumentos com números maiores que 10.

```
#!/bin/bash

for i in $@
do
  echo $i | grep "^[0-9]*$" > /dev/null
  if [ $? -eq 0 ]
  then
    if [ $i -gt 10 ]
    then
      echo -n "$i "
    fi
  fi
done
echo ""
```

Linux Professional Institute

Tópico 4: O Sistema Operacional Linux

 | Licenciado sob CC BY-NC-ND 4.0.

4.1 Escolhendo um Sistema Operacional

Referência ao LPI objectivo

Linux Essentials version 1.6, Exam 010, Objective 4.1

Peso

1

Áreas chave de conhecimento

- Diferenças entre Windows, Mac, Linux
- Gerenciamento do ciclo de vida das distribuições

Segue uma lista parcial de arquivos, termos e utilitários utilizados

- GUI versus linha de comando, configuração desktop
- Ciclos de manutenção, Beta e Stable

4.1 Lição 1

Certificação:	Linux Essentials
Versão:	1.6
Tópico:	4 O sistema operacional Linux
Objetivo:	4.1 A Escolha de um Sistema Operacional
Lição:	1 de 1

Introdução

Quer você use um computador em casa, na universidade ou dentro de uma empresa, ainda assim será necessário tomar uma decisão sobre o sistema operacional a ser usado. Claro que essa decisão pertence a você no caso do seu computador pessoal, mas talvez você também seja o responsável por fazer a escolha de um sistema para sua empresa. Como sempre, é essencial estar bem informado sobre as opções disponíveis para tomar uma decisão responsável. Nesta lição, vamos ajudá-lo a conhecer melhor as opções disponíveis de sistemas operacionais.

O que é um sistema operacional

Uma das primeiras coisas que precisamos deixar claro antes de começar nossa jornada pela escolha de um sistema operacional é entender o que queremos dizer com o termo. O sistema operacional reside no coração do seu computador e permite executar aplicativos dentro e por cima dele. Além disso, o sistema operacional contém drivers para acessar o hardware do computador, como discos e partições, telas, teclados, placas de rede e assim por diante. Costumamos abreviar o termo “sistema operacional” simplesmente como *SO*. Existem atualmente muitos sistemas operacionais para uso em computadores empresariais e também para uso pessoal e doméstico. Para simplificar a seleção disponível, podemos agrupar as opções da seguinte

maneira:

- Sistemas operacionais baseados em Linux
  - Linux para empresas
  - Linux para o usuário comum
- UNIX
- macOS
- Sistemas operacionais baseados em Windows
  - Servidores Windows
  - Sistemas desktop Windows

Escolhendo uma distribuição Linux

O kernel do Linux e as distribuições do Linux

Quando falamos em distribuições Linux, o sistema operacional é o Linux. O Linux é o *kernel*, ou seja, o cerne de toda distribuição Linux. O software do kernel do Linux é mantido por um grupo de indivíduos liderado por Linus Torvalds. Torvalds é empregado por um consórcio de empresas chamado The Linux Foundation para trabalhar no kernel do Linux.

NOTE O kernel do Linux foi desenvolvido inicialmente por Linus Torvalds, um estudante finlandês, em 1991. Em 1992, o primeiro lançamento do Kernel sob a GNU Licença Pública Geral versão 2 (GPLv2) foi a versão 0.12.

Kernel do Linux

Como dissemos, todas as distribuições Linux rodam o mesmo sistema operacional, Linux.

Distribuição Linux

Quando falamos de Red Hat Linux, ou Ubuntu Linux, referimo-nos à *distribuição Linux*. A distribuição Linux é fornecida com um kernel do Linux e um ambiente que nos permite interagir com ele. No mínimo, precisamos de um shell de linha de comando como o Bash e um conjunto de comandos básicos que nos permitam acessar e gerenciar o sistema. Claro que, na maioria das vezes, a distribuição Linux inclui um ambiente de desktop completo, como o Gnome ou o KDE.

Embora todas as distribuições Linux executem o sistema operacional Linux, as distribuições têm bastante variedade na versão do sistema operacional usada. Ou seja, *a versão do Kernel do Linux que está sendo usada* quando a distribuição é inicializada.

TIP

Se tiver acesso a uma linha de comando do Linux no momento, você pode conferir facilmente qual a versão do kernel do Linux que está usando lendo o *kernel release*:

```
$ uname -r
4.15.0-1019-aws
```

Tipos de distribuições Linux

Pode parecer uma escolha óbvia sempre executar a versão mais recente do kernel do Linux, mas não é tão simples assim. Podemos categorizar vagamente as distribuições do Linux em três conjuntos:

- Distribuições Linux de nível empresarial
  - Red Hat Enterprise Linux
  - CentOS
  - SUSE Linux Enterprise Server
  - Debian GNU/Linux
  - Ubuntu LTS
- Distribuições Linux para o usuário doméstico
  - Fedora
  - Ubuntu não-LTS
  - openSUSE
- Distribuições de Linux experimentais e para hackers
  - Arch
  - Gentoo

É claro que esse é apenas um subconjunto muito pequeno das distribuições possíveis, mas o mais importante é a diferença entre as distribuições *empresariais*, *de usuário* e *experimentais*, e por que cada uma delas existe.

Linux empresarial

Distribuições como o CentOS (*Community Enterprise OS*) são projetadas para serem implantadas em grandes organizações usando hardware corporativo. As necessidades corporativas são muito diferentes das necessidades de pequenas empresas, amadores ou usuários domésticos. Para garantir a disponibilidade de seus serviços, os usuários corporativos

têm requisitos mais elevados em relação à estabilidade de seu hardware e software. Portanto, as distribuições empresariais do Linux tendem a incluir versões mais antigas do kernel e de outros softwares, que funcionam de maneira confiável. Geralmente, as distribuições portam as atualizações importantes, como correções de segurança, para essas versões estáveis. Em troca, as distribuições corporativas do Linux podem não ter suporte para o hardware de consumo mais recente, além de conter versões mais antigas dos pacotes de software. No entanto, como no caso das distribuições Linux para usuários finais, as empresas tendem a escolher componentes de hardware maduros e desenvolver seus serviços em versões estáveis de software.

Linux para o usuário final

Distribuições como o Ubuntu são mais direcionadas para pequenas empresas ou particulares e usuários ocasionais. Esse tipo de usuário tem mais probabilidade de estar usando o hardware mais recente do mercado. Esses sistemas precisam dos drivers mais recentes para aproveitar ao máximo o novo hardware. Assim, o kernel mais recente é ideal para os usuários domésticos, mesmo tendo sido menos testado. Os kernels Linux mais recentes trazem os drivers mais atuais para suportar o hardware moderno que provavelmente estará em uso. Dado o crescimento do Linux que tem sido constatado no mercado de jogos, é importantíssimo que os drivers mais recentes estejam disponíveis para esses usuários.

NOTE

Algumas distribuições, como o Ubuntu, oferecem versões para o usuário doméstico, que contêm software recente e recebem atualizações por um curto período, além das chamadas versões com suporte a longo prazo (Long Term Support ou LTS), mais adequadas para ambientes empresariais.

Distribuições Linux experimentais e para hackers

Distribuições como o Arch Linux ou o Gentoo Linux vivem na vanguarda da tecnologia. Elas trazem as versões mais recentes do software, mesmo que essas versões ainda contenham erros e recursos não testados. Por outro lado, essas distribuições tendem a usar um modelo de lançamento contínuo (rolling release) que lhes permite entregar atualizações a qualquer momento. Essas distribuições são usadas por usuários avançados que desejam sempre ter acesso ao software mais recente, têm consciência de que a funcionalidade pode quebrar a qualquer momento e são capazes de reparar seu sistema quando isso acontece.

Em resumo, ao considerar o uso do Linux como sistema operacional, se você estiver usando hardware de nível corporativo em seus servidores ou computadores desktop, poderá escolher distribuições Linux de nível corporativo ou de usuário doméstico. Se você estiver usando hardware de nível doméstico e quiser aproveitar ao máximo as inovações de hardware mais recentes, provavelmente precisará de uma distribuição Linux igualmente recente para atender às necessidades do hardware.

Algumas distribuições Linux têm parentesco entre si. O Ubuntu, por exemplo, é baseado no Debian Linux e usa o mesmo sistema de empacotamento, DPKG. Já o Fedora é uma plataforma de teste para o RedHat Enterprise Linux, onde recursos potenciais de versões futuras do RHEL podem ser explorados antes de serem disponibilizados na distribuição corporativa.

Existem muitas outras distribuições Linux além das que mencionamos nesta lição. Uma vantagem de o Linux ser um software de código aberto é que muitas pessoas podem desenvolvê-lo da maneira como acham melhor. Graças a isso, existem centenas de distribuições. Para conhecer mais distribuições Linux, visite The Distro Watch Web Site. Os mantenedores desse site listam as 100 distribuições Linux mais baixadas, permitindo comparar e ver o que é popular atualmente.

Ciclo de vida do suporte no Linux

Como seria de se esperar, as distribuições corporativas do Linux têm uma vida útil mais longa do que as edições para o consumidor ou para a comunidade. Por exemplo, o Red Hat Enterprise Linux oferece suporte por 10 anos. O Red Hat Enterprise Linux 8 foi lançado em maio de 2019 e as atualizações e suporte ao software estarão disponíveis até maio de 2029.

As edições para o usuário doméstico geralmente só contam com o suporte da comunidade por meio de fóruns. As atualizações de software costumam ficar disponíveis por três lançamentos. Se tomarmos o Ubuntu como exemplo, no momento em que escrevemos este artigo a versão mais recente disponível era a 19.04, que foi atualizada no lançamento 19.10 e interrompida em janeiro de 2020. O Ubuntu também oferece edições com suporte a longo prazo, conhecidas como edições *LTS*, que contam com 5 anos de suporte a partir do lançamento original. A versão LTS atual é a 18.04, que terá atualizações de software até 2023. Essas versões LTS fazem do Ubuntu uma opção razoável para uso em empresas, graças ao suporte comercial disponibilizado pela Canonical (a empresa por trás da marca Ubuntu) ou por empresas de consultoria independentes.

NOTE As distribuições do Ubuntu usam datas como números de versão, no formato YY.MM: Por exemplo, a versão 19.04 foi lançada em abril de 2019.

O Linux no desktop

A implementação do Linux como sistema desktop pode ser mais difícil em uma empresa cuja equipe de suporte técnico está mais habituada a sistemas operacionais comerciais. No entanto, não é apenas o suporte que pode causar dificuldades. Um cliente corporativo também pode ter feito grandes investimentos em soluções de software que o prendem a sistemas operacionais específicos. Dito isso, há muitos exemplos de desktops Linux sendo integrados em grandes organizações; empresas como a Amazon, por exemplo, têm sua própria distribuição Linux Amazon Linux 2. Ela é usada na plataforma de nuvem da AWS, mas também internamente, em servidores e desktops.

Usar o Linux em uma pequena empresa ou em casa é bem mais fácil e pode ser uma experiência gratificante, eliminando a necessidade de pagar por uma licença e abrindo seus olhos para o verdadeiro tesouro de programas gratuitos e de código aberto disponíveis para o Linux. Você também descobrirá que existem muitos ambientes de desktop diferentes à disposição. Os mais comuns são o Gnome e o KDE, porém existem muitos outros. A decisão só depende de sua preferência pessoal.

Usando o Linux em servidores

No mundo empresarial, é muito comum que o Linux seja usado como sistema operacional nos servidores. Os servidores são mantidos por engenheiros especializados em Linux. Mesmo que haja milhares de usuários, eles não precisam saber nada sobre os servidores aos quais estão se conectando. O sistema operacional do servidor não é importante para eles e, em geral, os aplicativos clientes não diferem entre o Linux e outros sistemas operacionais no backend. Também é verdade que, à medida que mais aplicativos vão sendo virtualizados ou postos em contêineres dentro de nuvens locais e remotas, o sistema operacional vai ficando mais mascarado, e há grande probabilidade de o sistema operacional embarcado ser Linux.

Linux na nuvem

Outra oportunidade de se familiarizar com o Linux é implantá-lo em uma das muitas nuvens públicas disponíveis. A criação de uma conta em um dos muitos provedores de nuvem permite implantar rapidamente muitas distribuições diferentes do Linux de maneira rápida e fácil.

Sistemas operacionais não-Linux

Sim, por incrível que pareça, existem sistemas operacionais que não são baseados no kernel do Linux. É claro que, ao longo dos anos, existiram muitos e alguns deles caíram no esquecimento, mas ainda existem opções disponíveis para você, seja em casa ou no trabalho.

Unix

Antes de termos o Linux como sistema operacional, havia o Unix. O Unix costumava ser vendido junto com o hardware. Ainda hoje, diversos Unixes comerciais, como o AIX e o HP-UX, estão disponíveis no mercado. Embora o Linux tenha sido altamente inspirado no Unix (e na falta de disponibilidade dele para determinado hardware), a família de sistemas operacionais BSD baseia-se diretamente nele. Hoje, o FreeBSD, o NetBSD e o OpenBSD, junto com alguns outros sistemas BSD relacionados, estão disponíveis como software livre.

O Unix costumava ser muito usado no mundo empresarial, mas sofreu um declínio com o crescimento do Linux. À medida que o Linux cresceu e as ofertas de suporte corporativo também

foram aumentando, vimos o Unix lentamente começar a sumir. O Solaris, originalmente da Sun antes de se mudar para a Oracle, desapareceu recentemente. Ele era um dos maiores sistemas operacionais Unix usados pelas empresas de telecomunicações, anunciado como *Telco Grade Unix* (Unix em nível de operadora de telecomunicações).

Dentre os sistemas operacionais Linux, temos:

- AIX
- FreeBSD, NetBSD, OpenBSD
- HP-UX
- Irix
- Solaris

macOS

O macOS (anteriormente OS X) da Apple data de 2001. Largamente inspirado no BSD Unix e usando o shell de linha de comando Bash, trata-se um sistema amigável para quem está acostumado a usar sistemas operacionais Unix ou Linux. Se você estiver usando o macOS, pode abrir o aplicativo Terminal para acessar a linha de comando. Use o mesmo comando `uname` para verificar o sistema operacional relatado:

```
$ uname -s
Darwin
```

NOTE Neste caso, usamos a opção `-s` para retornar o nome do sistema operacional. Anteriormente, usamos `-r` para retornar o número da versão do kernel.

Microsoft Windows

Ainda podemos dizer que a maioria dos desktops e laptops em circulação estão rodando Windows. Esse sistema operacional conheceu um verdadeiro sucesso, dominando o mercado de desktops há anos. Embora se trate de um software proprietário e não seja gratuito, em geral a licença do sistema operacional é incluída na compra do hardware, facilitando a escolha do usuário. Obviamente, todos os fornecedores de hardware e software oferecem amplo suporte ao Windows, além de existirem muitos aplicativos de código aberto para o sistema. O futuro para o Windows não parece tão brilhante quanto no passado. Com a diminuição nas vendas de desktops e laptops, o foco passou a ser o mercado de tablets e celulares. Esse mercado foi dominado pela Apple e pelo Android, e a Microsoft está tendo dificuldades em ganhar terreno.

Como plataforma de servidor, a Microsoft agora permite que seus clientes escolham entre uma GUI (*Graphical User Interface*, ou interface gráfica de usuário) e uma versão apenas em linha de comando. A separação entre a GUI e a linha de comando é importante. Na maioria das vezes, a GUI dos Microsoft Servers mais antigos é carregada, mas ninguém a usa. Veja, por exemplo, o controlador de domínio do Active Directory... os usuários o usam o tempo todo para se autenticar no domínio, mas ele é gerenciado remotamente nas estações de trabalho dos administradores e não no servidor.

Exercícios Guiados

1. Qual projeto forma o componente comum de todas as distribuições Linux?

CentOS	
Red Hat	
Ubuntu	
Kernel do Linux	
CoreOS	

2. Qual sistema operacional é relatado como estando em uso no macOS da Apple?

OS X	
OSX	
Darwin	
MacOS	

3. Qual a diferença entre uma distribuição Linux e o kernel do Linux?

O kernel é parte de uma distribuição e a distribuição reúne aplicativos em volta do kernel, tornando-o útil	
O kernel é a distribuição Linux	
Todas as distribuições que usam o mesmo kernel são iguais	

4. Qual destes é um ambiente de desktop no Linux?

Mint	
Elementary	
Zorin	
Wayland	

5. Qual componente de um sistema operacional permite o acesso ao hardware?

Drivers	
Shells	
Serviços	
Aplicativos	

Exercícios Exploratórios

1. Se tiver acesso à linha de comando, encontre a lançamento atual do Kernel de seu sistema.

2. Usando seu mecanismo de pesquisa preferido, localize e identifique os provedores de nuvem pública disponíveis para você. Dentre eles existe o AWS, o Google Cloud, o Rackspace e muitos outros. Escolha um e veja quais sistemas operacionais estão disponíveis para implementação.

Resumo

Nesta seção, você aprendeu como diferenciar entre os diferentes sistemas operacionais mais comuns. Nós falamos de:

- Sistemas operacionais baseados em Linux
- UNIX
- macOS
- Sistemas operacionais baseados em Windows

Dentro da categoria Linux, podemos dividir ainda mais a seleção entre as distribuições com suporte a longo prazo e aquelas com um ciclo de suporte mais curto. As versões LTS são mais adequadas para empresas; o suporte de curto prazo é mais direcionado aos usuários domésticos e ocasionais.

- Distribuições Linux de nível empresarial
  - Red Hat Enterprise Linux
  - CentOS
  - SUSE Linux Enterprise Server
  - Debian GNU/Linux
  - Ubuntu LTS
- Distribuições Linux para o usuário doméstico
  - Fedora
  - Ubuntu não-LTS
  - openSUSE
- Distribuições de Linux experimentais e para hackers
  - Arch
  - Gentoo

Respostas aos Exercícios Guiados

1. Qual projeto forma o componente comum de todas as distribuições Linux?

CentOS	
Red Hat	
Ubuntu	
Kernel do Linux	X
CoreOS	

2. Qual sistema operacional é relatado como estando em uso no macOS da Apple?

OS X	
OSX	
Darwin	X
MacOS	

3. Qual a diferença entre uma distribuição Linux e o kernel do Linux?

O kernel é parte de uma distribuição, a distribuição reúne aplicativos em volta do kernel, tornando-o útil	X
O kernel é a distribuição Linux	
Todas as distribuições que usam o mesmo kernel são iguais	

4. Qual destes é um ambiente de desktop no Linux?

Mint	
Elementary	
Zorin	
Wayland	X

5. Qual componente de um sistema operacional permite o acesso ao hardware?

Drivers	X
Shells	
Serviços	
Aplicativos	

Respostas aos Exercícios Exploratórios

1. Se tiver acesso à linha de comando, encontre a lançamento atual do Kernel de seu sistema.

```
$ uname -r
4.15.0-47-generic
```

2. Usando seu mecanismo de pesquisa preferido, localize e identifique os provedores de nuvem pública disponíveis para você. Dentre eles existe o AWS, o Google Cloud, o Rackspace e muitos outros. Escolha um e veja quais sistemas operacionais estão disponíveis para implementação.

   O AWS, por exemplo, permite implementar várias distribuições Linux como Debian, Red Hat, SUSE ou Ubuntu, além do Windows.

4.2 Entendendo o Hardware do Computador

Referência ao LPI objectivo

Linux Essentials version 1.6, Exam 010, Objective 4.2

Peso

2

Áreas chave de conhecimento

- Hardware

Segue uma lista parcial de arquivos, termos e utilitários utilizados

- Placas-mãe, processadores, fontes, drives ópticos, periféricos
- Discos-rígidos e partições, /dev/sd*
- Drivers

4.2 Lição 1

Certificação:	Linux Essentials
Versão:	1.6
Tópico:	4 O sistema operacional Linux
Objetivo:	4.2 Entendendo o hardware do computador
Lição:	1 de 1

Introdução

Sem o hardware, o software nada mais é do que outra forma de literatura. O hardware processa os comandos descritos pelo software e fornece mecanismos para armazenamento, entrada e saída de dados. Até mesmo a nuvem está escorada em hardware.

Em seu papel como sistema operacional, uma das responsabilidades do Linux é fornecer software com interfaces que permitem acessar o hardware de um sistema. A maioria das especificações de configuração está além do escopo desta lição. No entanto, os usuários geralmente precisam se preocupar com o desempenho, a capacidade e outros fatores relacionados ao hardware, pois são eles que permitem que um sistema seja capaz de suportar adequadamente aplicativos específicos. Esta lição discute o hardware como itens físicos separados usando conectores e interfaces padrão. Os padrões são relativamente estáticos. Mas a forma, o desempenho e as características de capacidade do hardware estão em constante evolução. Mesmo que essas evoluções afetem as distinções físicas, os aspectos conceituais do hardware descritos nesta lição permanecem válidos.

NOTE

Em vários pontos desta lição, usamos exemplos na linha de comando para demonstrar maneiras de acessar informações sobre o hardware. A maioria dos exemplos vem de um Raspberry Pi B +, mas aplica-se à maioria dos sistemas. Não é

necessário entender esses comandos para compreender este material.

Fontes de alimentação

Todos os componentes ativos de um sistema de computação requerem eletricidade para operar. Infelizmente, a maioria das fontes de eletricidade não é apropriada. O hardware de um sistema de computação requer tensões elétricas específicas com tolerâncias relativamente estreitas. Não é bem o que a tomada da parede da sua casa pode oferecer.

As fontes de alimentação normalizam o fornecimento de energia disponível. Os requisitos de tensão padronizados permitem que os fabricantes criem componentes de hardware que podem ser usados em sistemas de qualquer lugar do mundo. As fontes de alimentação de computadores pessoais costumam usar a eletricidade das tomadas como suprimento. As fontes de alimentação de um servidor são cruciais para seu funcionamento, e portanto costumam ser conectadas a várias fontes de fornecimento para garantir que continuem operando se uma delas falhar.

O consumo de energia gera calor. O calor excessivo pode fazer com que os componentes do sistema operem lentamente ou até parem de funcionar. A maioria dos sistemas possui algum tipo de ventoinha para promover um resfriamento mais eficiente. Componentes como os processadores geralmente produzem um calor que o fluxo de ar por si só não é capaz de dissipar. Esses componentes quentes vêm com aletas especiais, conhecidas como dissipadores de calor, para ajudar a expulsar o calor que geram. Os dissipadores de calor geralmente têm sua própria ventoinha para garantir um fluxo de ar adequado.

Placa-mãe

Todo o hardware de um sistema precisa ser interconectado. A placa-mãe normaliza essa interconexão usando conectores padronizados e fatores de forma. Ela também fornece suporte para a configuração e as necessidades elétricas desses conectores.

Existem muitas configurações possíveis para a placa-mãe. Elas suportam diferentes processadores e sistemas de memória, trazem diferentes combinações de conectores padronizados e se adaptam aos diferentes tamanhos de acondicionamento. Tirando, talvez, a capacidade de conectar dispositivos externos específicos, a configuração da placa-mãe não tem muito segredo para os usuários. Os administradores do sistema precisam entender a configuração da placa-mãe quando é necessário identificar dispositivos específicos.

Logo que uma fonte de energia é aplicada pela primeira vez, é necessário configurar e inicializar um hardware específico à placa-mãe antes que o sistema possa operar. Para isso, as placas-mãe usam a programação armazenada na memória não-volátil, conhecida como firmware. A forma original do firmware da placa-mãe era conhecida como BIOS (*Basic Input/Output System*, ou

Sistema Básico de Entrada/Saída). Além das definições básicas de configuração, a BIOS era a principal responsável pela identificação, carregamento e transferência da operação para um sistema operacional, como o Linux. À medida que o hardware foi evoluindo, o firmware foi expandido para suportar discos maiores, diagnósticos, interfaces gráficas, rede e outros recursos avançados, independentemente do sistema operacional carregado. As primeiras tentativas de fazer avançar o firmware para além da BIOS básica eram frequentemente específicas a um fabricante de placas-mãe. A Intel definiu o padrão EFI (*Extensible Firmware Interface*) para o firmware avançado. Mais tarde, a empresa forneceu a EFI para um organismo de normalização como base para a criação da UEFI (*Unified Extensible Firmware Interface*). Hoje, a maioria das placas-mãe usa UEFI. É raro encontrar BIOS ou EFI em sistemas recentes. Apesar disso, a maioria das pessoas ainda se refere ao firmware da placa-mãe como BIOS.

Pouquíssimas configurações de firmware são de interesse para os usuários comuns, e assim, geralmente apenas os responsáveis pela configuração do hardware do sistema precisam lidar com o firmware e suas configurações. Uma das poucas opções comumente alteradas é a ativação das extensões de virtualização nas CPUs modernas.

Memória

A memória do sistema conserva os dados e o código do programa dos aplicativos em execução no momento. Quando falamos na memória do computador, em geral nos referimos a essa memória do sistema. Outro termo comum usado para a memória do sistema é a sigla RAM (*Random Access Memory*) ou alguma variação dela. Às vezes, também são usadas referências ao acondicionamento físico da memória do sistema, como DIMM, SIMM ou DDR.

Fisicamente, a memória do sistema costuma ser acondicionada em módulos de placas de circuito individuais que são conectados à placa-mãe. Atualmente, os módulos de memória individuais variam de 2 GB a 64 GB. Para a maioria das aplicações de uso geral, 4 GB é o mínimo de memória do sistema que os usuários devem considerar. Para estações de trabalho individuais, 16 GB normalmente são mais do que suficientes. No entanto, mesmo 16 GB podem ser limitantes para usuários que executam jogos, vídeos ou aplicativos de áudio de última geração. Os servidores geralmente exigem 128 GB ou até 256 GB de memória para oferecer um suporte eficiente a todos os usuários.

Na maioria dos casos, o Linux permite que os usuários tratem a memória do sistema como uma caixa preta. Um aplicativo é iniciado e o Linux trata de alocar a memória do sistema necessária a ele. Quando o aplicativo é encerrado, o Linux libera a memória para uso de outros aplicativos. Mas e se um aplicativo exigir mais do que a memória do sistema disponível? Nesse caso, o Linux move os aplicativos inativos da memória do sistema para uma área especial do disco conhecida como espaço de troca (swap space). O Linux move os aplicativos inativos do espaço de troca de volta para a memória do sistema quando eles precisam ser executados.

Os sistemas sem hardware de vídeo dedicado costumam usar uma parte da memória do sistema (geralmente 1 GB) como armazenamento para exibição de vídeo. Isso reduz a memória efetiva do sistema. O hardware de vídeo dedicado possui normalmente sua própria memória separada, que não está disponível como memória do sistema.

Existem várias maneiras de obter informações sobre a memória do sistema. Para o usuário, os valores mais interessantes são a quantidade total de memória disponível e em uso. Uma fonte de informação seria executar o comando `free` junto com o parâmetro `-m` para gerar uma saída em megabytes:

```
$ free -m
              total        used        free      shared  buff/cache   available
Mem:            748          37          51          14         660         645
Swap:            99           0          99
```

A primeira linha especifica a memória total disponível para o sistema (`total`), a memória em uso (`used`) e a memória livre (`free`). A segunda linha exibe as informações correspondentes ao espaço de troca. A memória indicada como `shared` e `buff/cache` está sendo empregada atualmente para outras funções do sistema, embora a quantidade indicada em `available` possa ser usada para aplicação.

Processadores

A palavra “processador” implica que algo está sendo processado. Nos computadores, a maior parte desse processamento refere-se a sinais elétricos. Normalmente, esses sinais são tratados como tendo um dos valores binários de 1 ou 0.

Quando falamos em computadores, costumamos usar o termo processador de forma intercambiável com a sigla CPU (*Central Processing Unit*, ou Unidade de Processamento Central). Isso não é tecnicamente correto. Todo computador de uso geral possui uma CPU que processa os comandos binários especificados pelo software. Portanto, é compreensível que as pessoas confundam processador e CPU. No entanto, além de uma CPU, os computadores modernos geralmente incluem outros processadores de tarefas específicos. O processador adicional mais conhecido é provavelmente a GPU (*Graphical Processing Unit*, ou Unidade de Processamento Gráfico). Assim, embora uma CPU seja um processador, nem todos os processadores são CPUs.

Para a maioria das pessoas, a arquitetura da CPU é uma referência às instruções que o processador suporta. Embora a Intel e a AMD produzam processadores que suportam as mesmas instruções, é importante diferenciá-los por fornecedor devido às diferenças específicas de acondicionamento, desempenho e consumo de energia. As distribuições de software geralmente

usam as seguintes designações para especificar o conjunto mínimo de instruções necessárias para operar:

i386

Referencia o conjunto de instruções de 32 bits associado ao Intel 80386.

x86

Referencia tipicamente os conjuntos de instruções de 32 bits associados aos sucessores do 80386, como 80486, 80586 e Pentium.

x64 / x86-64

Referencia os processadores que suportam as instruções de 32 e 64 bits da família x86.

AMD

Uma referência ao suporte do x86 pelos processadores AMD.

AMD64

Uma referência ao suporte do x64 pelos processadores AMD.

ARM

Referencia uma CPU *Reduced Instruction Set Computer* (RISC) que não se baseia no conjunto de instruções x86. Geralmente usado por dispositivos embarcados e móveis, tablets e dispositivos operados por bateria. Uma versão do Linux para ARM é usada pelo Raspberry Pi.

O arquivo `/proc/cpuinfo` contém informações detalhadas sobre o processador de um sistema. Infelizmente, os detalhes não são amigáveis para os usuários comuns. Um resultado mais geral pode ser obtido com o comando `lscpu`. Eis a saída de um Raspberry Pi B +:

```
$ lscpu
Architecture:          armv7l
Byte Order:            Little Endian
CPU(s):                4
On-line CPU(s) list:   0-3
Thread(s) per core:    1
Core(s) per socket:    4
Socket(s):             1
Model:                 4
Model name:            ARMv7 Processor rev 4 (v7l)
CPU max MHz:           1400.0000
CPU min MHz:           600.0000
BogoMIPS:              38.40
Flags:                 half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva idivt vfpd32
```

```
lpae evtstrm crc32
```

Para a maioria das pessoas, a infinidade de fornecedores, famílias de processadores e fatores de especificação representam uma variedade desconcertante de opções. Apesar disso, há vários fatores associados a CPUs e processadores que mesmo os usuários comuns e administradores precisam levar em consideração quando precisam especificar ambientes operacionais:

Tamanho do bit

Para as CPUs, esse número refere-se ao tamanho nativo dos dados que ele manipula e à quantidade de memória que pode acessar. A maioria dos sistemas modernos são de 32 ou 64 bits. Se um aplicativo precisar acessar mais de 4 gigabytes de memória, ele deverá ser executado em um sistema de 64 bits, pois 4 gigabytes é o número máximo que pode ser representado usando 32 bits. E, embora aplicativos de 32 bits normalmente possam ser executados em sistemas de 64 bits, aplicativos de 64 bits não podem ser executados em sistemas de 32 bits.

Velocidade de clock

Geralmente expressa em megahertz (MHz) ou gigahertz (GHz). Relaciona-se à rapidez com que um processador processa instruções. Mas a velocidade do processador é apenas um dos fatores que afetam os tempos de resposta, tempos de espera e taxa de transferência do sistema. Mesmo ao realizarmos várias tarefas ao mesmo tempo, é raro que a CPU de um PC de mesa comum esteja ativa mais de 2 ou 3% do tempo. Ainda assim, se você costuma lançar aplicativos exigentes, que envolvem atividades como criptografia ou renderização de vídeo, a velocidade da CPU pode ter um impacto significativo na taxa de transferência e no tempo de espera.

Cache

As CPUs exigem um fluxo constante de instruções e dados para operar. O custo e o consumo de energia de uma memória de sistema rica em gigabytes que pudesse ser acessada na velocidade do clock da CPU seriam proibitivos. A memória cache da CPU é integrada ao chip do processador e fornece um buffer de alta velocidade entre as CPUs e a memória do sistema. A cache é separada em várias camadas, comumente chamadas de L1, L2, L3 e até L4. Em geral, no caso da cache, quanto mais, melhor.

Núcleos

Chamamos de núcleo uma CPU individual. Além do núcleo que representa uma CPU física, a *Hyper-Threading Technology* (HTT) permite que uma única CPU física processe simultaneamente múltiplas instruções, atuando virtualmente como várias CPUs físicas. Na maioria dos casos, múltiplos núcleos físicos são empacotados como um único processador físico. No entanto, existem placas-mãe que suportam múltiplos processadores físicos. Em teoria, um número maior de núcleos deveria proporcionar um melhor rendimento do sistema.

Infelizmente, os aplicativos de desktop geralmente mantêm as CPUs ocupadas 2 ou 3% do tempo e, portanto, adicionar mais CPUs ociosas provavelmente resultaria em uma melhoria mínima na taxa de transferência. É mais adequado dispor de mais núcleos para a execução de aplicativos que são criados para ter vários segmentos de operação independentes, como renderização de frames de vídeo, renderização de página web ou ambientes de máquina virtual multiusuário.

Armazenamento

Os dispositivos de armazenamento são um método de preservar programas e dados. As *Unidades de Disco Rígido* (Hard Disk Drives ou HDDs) e as *Unidades de Estado Sólido* (Solid State Drives ou SSDs) são a forma mais comum de dispositivo de armazenamento para servidores e desktops. Também usamos cartões de memória USB e dispositivos óticos, como os DVDs, mas raramente como dispositivo primário.

Como o nome indica, uma unidade de disco rígido armazena informações em um ou mais discos rígidos físicos. Os discos físicos são cobertos com um material magnético que possibilita o armazenamento. Os discos estão contidos em um acondicionamento lacrado, pois a poeira, pequenas partículas e até impressões digitais interferem na capacidade do HDD de ler e gravar na mídia magnética.

Os SSDs são, grosso modo, versões mais sofisticadas de dispositivos de memória USB com capacidade significativamente maior. Os SSDs armazenam informações em microchips em que não há partes móveis.

Embora a tecnologia por trás dos HDDs e SSDs seja diferente, é possível compará-los em uma série de fatores importantes. A capacidade do disco rígido baseia-se no dimensionamento dos componentes físicos, enquanto a capacidade do SSD depende do número de microchips. Os SSDs custam entre 3 e 10 vezes mais que um HDD por gigabyte. Para ler ou gravar, um HDD deve esperar que um ponto do disco gire até um local conhecido, ao passo que os SSDs são de acesso aleatório. As velocidades de acesso do SSD são tipicamente de 3 a 5 vezes maiores que nos dispositivos HDD. Como não têm partes móveis, os SSDs consomem menos energia e são mais confiáveis do que os HDDs.

A capacidade de armazenamento dos HDDs e SSDs está aumentando constantemente. Atualmente, HDDs de 5 terabytes e SSDs de 1 terabyte são facilmente encontrados no mercado. Apesar disso, uma maior capacidade de armazenamento nem sempre é melhor. Quando um dispositivo de armazenamento falha, todas as informações são perdidas. E, é claro, o backup leva mais tempo quando há mais informações para salvar. Para aplicativos que leem e gravam uma grande quantidade de dados, a latência e o desempenho podem ser mais importantes do que a capacidade.

Os sistemas modernos usam SCSI (*Small Computer System Interface*) ou SATA (*Serial AT Attachment*) para conectar-se aos dispositivos de armazenamento. Essas interfaces são tipicamente suportadas pelo conector apropriado na placa-mãe. O carregamento inicial do sistema vem de um dispositivo de armazenamento conectado à placa-mãe. As configurações de firmware definem a ordem em que os dispositivos são acessados para esse carregamento inicial.

Os sistemas de armazenamento conhecidos como RAID (*Redundant Array of Independent Disks*) são uma implementação comum para evitar a perda de informações. Uma matriz RAID consiste em vários dispositivos físicos que contêm cópias duplicadas das informações. Se um desses dispositivos falhar, todas as informações ainda estarão disponíveis nos outros. As diferentes configurações físicas de RAID são conhecidas como 0, 1, 5, 6 e 10. Cada designação oferece diferentes tamanhos de armazenamento, características de desempenho e maneiras de armazenar dados redundantes ou somas de verificação para recuperação de dados. Tirando um pequeno trabalho extra de configuração do administrador, a existência de um RAID passa despercebida para os usuários.

Os dispositivos de armazenamento geralmente leem e gravam dados na forma de blocos de bytes. O comando `lsblk` pode ser usado para listar os dispositivos de bloco disponíveis para um sistema. O exemplo a seguir foi executado em um Raspberry Pi usando um cartão SD como dispositivo de armazenamento. Os detalhes da saída serão abordado nas lições sobre *Partições* e *Drivers* a seguir:

```
$ lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
mmcblk0     179:0    0 29.7G  0 disk
+-mmcblk0p1 179:1    0 43.9M  0 part /boot
+-mmcblk0p2 179:2    0 29.7G  0 part /
```

Partições

Um dispositivo de armazenamento é, na verdade uma longa sequência de locais de armazenamento. O particionamento é o mecanismo que informa ao Linux se ele deve tratar esses locais de armazenamento como uma única sequência ou como diversas sequências independentes. Cada partição é tratada como se fosse um dispositivo individual. Na maioria das vezes, as partições são criadas quando um sistema é configurado pela primeira vez. Se for preciso mudar alguma coisa, existem ferramentas administrativas que permitem gerenciar o particionamento de dispositivos.

Mas qual seria o interesse de se configurar diversas partições? Alguns usos comuns de partições incluem gerenciar o espaço de armazenamento disponível, isolar a sobrecarga de criptografia ou

oferecer suporte a diferentes sistemas de arquivos. As partições possibilitam inicializar um único dispositivo de armazenamento em diferentes sistemas operacionais.

Embora o Linux seja capaz de reconhecer a sequência de armazenamento de um dispositivo bruto, um dispositivo bruto não pode ser usado do jeito que chega da loja. Antes, ele deve ser formatado. A formatação determina um sistema de arquivos para o dispositivo e o prepara para as operações de arquivo. Sem um sistema de arquivos, um dispositivo não pode ser usado para operações relacionadas a arquivos.

Os usuários veem as partições como se fossem dispositivos individuais. Dessa maneira, é fácil esquecer que só existe um único dispositivo físico. Particularmente, as operações de dispositivo para dispositivo - que na verdade são operações de partição para partição - não terão o desempenho esperado. Um único dispositivo é um mecanismo físico com um conjunto de hardware de leitura/gravação. Mas o mais importante a se lembrar é que não é possível usar as partições de um único dispositivo físico como um design à prova de falhas. Se o dispositivo falhar, todas as partições falharão ao mesmo tempo.

NOTE O *Gerenciador de Volume Lógico* (Logical Volume Manager ou LVM) é um recurso de software que permite aos administradores combinar discos individuais e partições de disco e tratá-los como uma única unidade.

Periféricos

Servidores e estações de trabalho precisam de uma combinação de CPU, memória do sistema e armazenamento para operar. Mas esses componentes fundamentais não se relacionam diretamente com o mundo externo. Os periféricos são os dispositivos que fornecem aos sistemas entradas, saídas e acesso ao resto do mundo real.

A maioria das placas-mãe possui conectores externos integrados e suporte em firmware para interfaces periféricas legadas comuns, suportando dispositivos como teclado, mouse, som, vídeo e rede. As placas-mãe recentes geralmente incluem um conector Ethernet para as redes, um conector HDMI para atender às necessidades gráficas básicas e um ou mais conectores USB (*Universal Serial Bus*) para quase todo o resto. Existem várias versões do USB, com velocidades e características físicas diferentes. É comum haver diversas versões de portas USB em uma única placa-mãe.

As placas-mãe também podem ter um ou mais slots de expansão. Os slots de expansão permitem que os usuários adicionem placas de circuito especiais, conhecidas como placas de expansão, que suportam periféricos personalizados, legados e não-padrão. As interfaces gráficas, de som e de rede são placas de expansão comuns. As placas de expansão também suportam RAID e interfaces legadas de formato especial envolvendo conexões seriais e paralelas.

As configurações do tipo *System on a Chip* (SoC) são mais vantajosas no consumo de energia, desempenho, espaço e confiabilidade em comparação com as configurações de placa-mãe, por reunirem processadores, memória do sistema, SSD e hardware para controlar periféricos em um único pacote de circuitos integrados. Os periféricos suportados pelas configurações de SoC são limitados pelos componentes incluídos. Assim, as configurações de SoC tendem a ser desenvolvidas para usos específicos. Telefones, tablets e outros dispositivos portáteis geralmente são baseados na tecnologia SoC.

Alguns sistemas já vêm com periféricos incorporados. Os laptops são semelhantes às estações de trabalho, mas incluem periféricos de exibição, teclado e mouse por padrão. Os sistemas Tudo-Em-Um são semelhantes aos laptops, mas requerem periféricos de mouse e teclado. Em geral os controladores da placa-mãe ou do SoC são fornecidos com periféricos integrados apropriados para um uso específico.

Drivers e arquivos de dispositivo

Até agora, esta lição apresentou informações sobre processadores, memória, discos, particionamento, formatação e periféricos. Mas se os usuários comuns precisassem lidar com os detalhes específicos de cada um dos dispositivos que decidissem conectar à sua máquina, esses sistemas seriam praticamente inutilizáveis. Da mesma forma, os desenvolvedores de software precisariam modificar seu código para cada dispositivo novo ou modificado que precisassem suportar.

A solução para este problema de “lidar com os detalhes” existe na forma de um driver de dispositivo. Os drivers de dispositivo aceitam um conjunto padrão de solicitações, convertendo essas solicitações nas atividades de controle apropriadas ao dispositivo. Os drivers de dispositivo permitem que você e os aplicativos executados leiam o arquivo `/home/carol/stuff` sem se preocupar em saber se esse arquivo está em um disco rígido, uma unidade de estado sólido, um cartão de memória, um armazenamento criptografado ou outro dispositivo .

Os arquivos de dispositivos são encontrados no diretório `/dev` e identificam os dispositivos físicos, o acesso aos dispositivos e os drivers suportados. Por convenção, nos sistemas modernos que usam dispositivos de armazenamento baseados em SCSI ou SATA, o nome do arquivo de especificação começa com o prefixo `sd`. O prefixo é seguido por uma letra como `a` ou `b`, indicando um dispositivo físico. Após o prefixo e o identificador do dispositivo, vem um número que indica uma partição no dispositivo físico. Portanto, `/dev/sda` se refere ao primeiro dispositivo de armazenamento inteiro, enquanto `/dev/sda3` se refere à partição 3 do primeiro dispositivo de armazenamento. Há uma convenção de nomenclatura apropriada ao arquivo do dispositivo para cada tipo de dispositivo. Embora não seja possível tratar de todas as convenções de nomenclatura possíveis nesta lição, é importante lembrar que essas convenções são essenciais para a administração do sistema.

O conteúdo do diretório `/dev` está além do escopo desta lição, mas ainda assim é informativo olhar para a entrada referente a um dispositivo de armazenamento. Os arquivos de dispositivos para cartões SD tipicamente usam `mmcblk` como prefixo:

```
$ ls -l mmcblk*
brw-rw---- 1 root disk 179, 0 Jun 30 01:17 mmcblk0
brw-rw---- 1 root disk 179, 1 Jun 30 01:17 mmcblk0p1
brw-rw---- 1 root disk 179, 2 Jun 30 01:17 mmcblk0p2
```

Os detalhes da lista para um arquivo de dispositivo são diferentes dos detalhes típicos de um arquivo:

- Ao contrário de um arquivo ou diretório, a primeira letra do campo de permissões é `b`. Isso indica que os blocos são lidos e gravados no dispositivo em blocos, e não em caracteres individuais.
- O campo tamanho tem dois valores separados por vírgula, em vez de um único valor. O primeiro valor geralmente indica um driver específico dentro do kernel e o segundo valor especifica um dispositivo específico operado pelo driver.
- O nome do arquivo usa um número para o dispositivo físico, portanto a convenção de nomenclatura se adapta especificando o sufixo da partição como um `p` seguido de um dígito.

NOTE

Cada dispositivo do sistema deve ter uma entrada em `/dev`. Como o conteúdo do diretório `/dev` é criado na instalação, costumam existir entradas para todos os drivers e dispositivos possíveis, mesmo que não haja nenhum dispositivo físico presente.

Exercícios Guiados

1. Descreva estes termos:

Processador	
CPU	
GPU	

2. Ao executar principalmente aplicativos de edição de vídeo (uma atividade intensiva em termos de computação), quais componentes e características normalmente teriam o maior impacto na usabilidade do sistema:

Núcleos da CPU	
Velocidade da CPU	
Memória disponível do sistema	
Sistema de armazenamento	
GPU	
Exibição de vídeo	
Nenhuma das anteriores	

3. Qual seria o nome do arquivo de dispositivo `/dev` para a partição 3 do terceiro drive SATA de um sistema:

`sd3p3`	
`sdcp3`	
`sdc3`	
Nenhum dos anteriores	

Exercícios Exploratórios

1. Rode o comando `lsblk` em seu sistema. Identifique os parâmetros abaixo. Caso não tenha um sistema à disposição, analise a lista `lsblk -f` do sistema Raspberry Pi mencionada na seção "Armazenamento", acima:

```
$ lsblk -f
NAME        FSTYPE LABEL  UUID                                 MOUNTPOINT
mmcblk0
+-mmcblk0p1 vfat   boot   9304-D9FD                            /boot
+-mmcblk0p2 ext4   rootfs 29075e46-f0d4-44e2-a9e7-55ac02d6e6cc /
```

   - Tipos de dispositivo e quantos são
   - A estrutura de partições de cada dispositivo
   - O tipo de sistema de arquivos e a montagem de cada partição

Resumo

Um sistema é a soma de seus componentes. Diferentes componentes afetam o custo, o desempenho e a usabilidade de diferentes maneiras. Embora existam configurações comuns para estações de trabalho e servidores, não há uma configuração que seja melhor que as outras.

Respostas aos Exercícios Guiados

1. Descreva estes termos:

   Processador

   Um termo geral que se aplica a qualquer tipo de processador. Muitas vezes usado incorretamente como sinônimo de CPU.

   CPU

   Unidade Central de Processamento. Uma unidade de processamento que suporta as tarefas computacionais de uso geral.

   GPU

   Unidade de Processamento Gráfico. Uma unidade de processamento otimizada para suportar atividades relacionadas à apresentação de imagens.

2. Ao executar principalmente aplicativos de edição de vídeo (uma atividade intensiva em termos de computação), quais componentes e características normalmente teriam o maior impacto na usabilidade do sistema:

   Núcleos da CPU

   Sim. Uma configuração com múltiplos núcleos suporta as tarefas concorrentes de apresentação e renderização exigidas pela edição de vídeo.

   Velocidade da CPU

   Sim. A renderização de vídeo requer uma dose significativa de atividade computacional.

   Memória disponível do sistema

   Provavelmente. O vídeo não-compactado usado na edição é grande. Os sistemas de uso geral costumam vir com 8 gigabytes de memória. Uma memória de 16 ou mesmo 32 gigabytes permitiria que o sistema operasse mais quadros de vídeo não compactado, aumentando a eficácia das atividades de edição.

   Sistema de armazenamento

   Sim. Os arquivos de vídeo são grandes. A sobrecarga de um drive SSD local suporta uma transferência mais eficiente. Drives de rede mais lentos seriam contraproducentes.

   GPU

   Não. A GPU teria um impacto principalmente na apresentação do vídeo renderizado.

Exibição de vídeo

Não. A exibição de vídeo teria um impacto principalmente na apresentação do vídeo renderizado.

Nenhuma das anteriores

Não. Alguns desses fatores têm um impacto óbvio na usabilidade do sistema no contexto apresentado.

3. Qual seria o nome do arquivo de dispositivo `/dev` para a partição 3 do terceiro drive SATA de um sistema:

`sd3p3`	Incorreto. O Drive 3 seria `sdc` e não `sd3`
`sdcp3`	Incorreto. A Partição 3 seria `3` e não `p3`
`sdc3`	Correto
Nenhum dos anteriores	Incorreto. Uma das opções traz a resposta correta.

Respostas aos Exercícios Exploratórios

1. Rode o comando `lsblk` em seu sistema. Identifique os parâmetros abaixo. Caso não tenha um sistema à disposição, analise a lista `lsblk -f` do sistema Raspberry Pi mencionada na seção "Armazenamento", acima:

```
$ lsblk -f
NAME        FSTYPE LABEL  UUID                                 MOUNTPOINT
mmcblk0
+-mmcblk0p1 vfat   boot   9304-D9FD                            /boot
+-mmcblk0p2 ext4   rootfs 29075e46-f0d4-44e2-a9e7-55ac02d6e6cc /
```

   As respostas a seguir baseiam-se na lista `lsblk -f` do sistema Raspberry Pi mostrado anteriormente. Suas respostas podem ser diferentes:

   Tipos de dispositivo e quantos são

   Há um só dispositivo: `mmcblk0`. Por convenção, sabemos que `mmcblk` seria um cartão de memória USB.

   A estrutura de partições de cada dispositivo

   Existem duas partições: `mmcblk0p1` e `mmcblk0p2`.

   O tipo de sistema de arquivos e a montagem de cada partição

   A Partição 1 usa o sistema de arquivos `vfat`. Ele é usado para inicializar o sistema e está montado como `/boot`. A Partição 2 usa o sistema de arquivos `ext4`. Ele é usado como o sistema de arquivos principais e está montado como `/`.

4.3 Onde os dados são armazenados

Referência ao LPI objectivo

Linux Essentials version 1.6, Exam 010, Objective 4.3

Peso

3

Áreas chave de conhecimento

- Programas e configuração, pacotes e banco de dados de pacotes
- Processos, endereços de memória, mensagens do sistema; logando-se.

Segue uma lista parcial de arquivos, termos e utilitários utilizados

- `ps`, `top`, `free`
- `syslog`, `dmesg`
- `/etc/`, `/var/log/`
- `/boot/`, `/proc/`, `/dev/`, `/sys/`

4.3 Lição 1

Certificação:	Linux Essentials
Versão:	1.6
Tópico:	4 O sistema operacional Linux
Objetivo:	4.3 Onde os dados são armazenados
Lição:	1 de 2

Introdução

Um sistema operacional considera que tudo são dados. Para o Linux, tudo é considerado como um arquivo: programas, arquivos regulares, diretórios, dispositivos de bloco (como os discos rígidos), dispositivos de caracteres (como os consoles), processos do kernel, sockets, partições, links etc. A estrutura de diretórios do Linux , começando com o *root* `/`, é uma coleção de arquivos que contêm dados. Essa redução de tudo a arquivos é um recurso poderoso do Linux, pois permite refinar praticamente todos os aspectos do sistema.

Nesta lição, discutiremos os diferentes locais em que dados importantes são armazenados, conforme estabelecido pelo Linux Filesystem Hierarchy Standard (FHS). Alguns desses locais são diretórios reais que armazenam dados de maneira permanente em discos, enquanto outros são pseudo-sistemas de arquivos carregados na memória que nos dão acesso aos dados do subsistema do kernel, como processos em execução, uso de memória, configuração de hardware e assim por diante. Os dados armazenados nesses diretórios virtuais são usados por uma série de comandos que nos permitem monitorar e manipular esses dados.

Programas e suas configurações

Sem a menor dúvida, os dados importantes de um sistema Linux são seus programas e os arquivos de configuração correspondentes. Os primeiros são arquivos executáveis que contêm conjuntos de instruções a serem executadas pelo processador do computador, ao passo que os últimos geralmente consistem em documentos de texto que controlam a operação de um programa. Os arquivos executáveis podem ser arquivos binários ou de texto. Os arquivos de texto executáveis são chamados de scripts. Os dados de configuração no Linux também são tradicionalmente armazenados em arquivos de texto, embora existam diversos estilos de representação de dados de configuração.

Onde os arquivos binários são armazenados

Como qualquer outro arquivo, os arquivos executáveis vivem em diretórios vinculados a `/`. Mais especificamente, os programas são distribuídos em uma estrutura de três camadas: a primeira camada (`/`) inclui os programas que podem ser necessários no modo de usuário único, a segunda camada (`/usr`) contém a maioria dos programas multiusuários e a terceira camada (`/usr/local`) é usada para armazenar o software que não é fornecido pela distribuição e foi compilado localmente.

Tipicamente, os programas estão nestes locais:

`/sbin`

Contém binários essenciais para a administração do sistema, como `parted` ou `ip`.

`/bin`

Contém binários essenciais para todos os usuários, como `ls`, `mv` ou `mkdir`.

`/usr/sbin`

Armazena binários para administração do sistema, como `deluser` ou `groupadd`.

`/usr/bin`

Inclui a maioria dos arquivos executáveis - como `free`, `pstree`, `sudo` or `man` — que podem ser acessados por todos os usuários.

`/usr/local/sbin`

Usado para armazenar programas instalados localmente para a administração do sistema e que não são gerenciados pelo gerenciador de pacotes do sistema.

`/usr/local/bin`

Tem a mesma finalidade de `/usr/local/sbin`, mas para programas comuns de usuário.

Recentemente, algumas distribuições começaram a substituir `/bin` e `/sbin` por links simbólicos para `/usr/bin` e `/usr/sbin`.

NOTE O diretório `/opt` às vezes é usado para armazenar aplicativos opcionais de terceiros.

Além desses diretórios, os usuários comuns podem ter seus próprios programas em:

- `/home/$USER/bin`
- `/home/$USER/.local/bin`

TIP Para descobrir quais diretórios estão disponíveis para executar binários, referencie a variável `PATH` com `echo $PATH`. Para saber mais sobre o `PATH`, reveja as lições sobre variáveis e customização do shell.

Podemos encontrar o local dos programas com o comando `which`:

```
$ which git
/usr/bin/git
```

Onde os arquivos de configuração são armazenados

O Diretório `/etc`

Nos primeiros dias do Unix, havia uma pasta para cada tipo de dados, como `/bin` para binários e `/boot` para o(s) kernel(s). No entanto, `/etc` (que significa literalmente *et cetera*) foi criado como um diretório abrangente para armazenar todos os arquivos que não pertenciam às outras categorias. A maioria deles eram arquivos de configuração. Com o passar do tempo, mais e mais arquivos de configuração foram adicionados, de modo que o `/etc` se tornou a pasta principal dos arquivos de configuração dos programas. Como já explicamos, um arquivo de configuração geralmente é um arquivo local de texto puro sem formatação (diferente de um arquivo binário) que controla a operação de um programa.

No `/etc`, encontramos diferentes padrões para os nomes dos arquivos de configuração:

- Arquivos com extensão *ad hoc* ou sem extensão, como por exemplo

  `group`
  : Banco de dados de grupo do sistema.

`hostname`

Nome do computador na rede.

`hosts`

Lista de endereços IP com sua tradução em nomes de computadores na rede.

`passwd`

Banco de dados do usuário do sistema — composto por sete campos separados por dois pontos, fornecendo informações sobre o usuário.

`profile`

Arquivo de configuração de todo o sistema para o Bash.

`shadow`

Arquivo criptografado para as senhas de usuário.

- Arquivos de inicialização terminados em `rc`:

`bash.bashrc`

Arquivo `.bashrc` de todo o sistema para shells bash interativos.

`nanorc`

Exemplo de arquivo de inicialização para o GNU nano (um editor de texto simples que normalmente é fornecido com qualquer distribuição).

- Arquivos terminados em `.conf`:

`resolv.conf`

Arquivo de configuração do resolvedor — que fornece acesso ao DNS (Internet Domain Name System).

`sysctl.conf`

Arquivo de configuração para definir variáveis de sistema para o kernel.

- Diretórios com sufixo `.d`:

Alguns programas com um arquivo de configuração exclusivo (`*.conf` ou outro) trazem um diretório `*.d` dedicado que ajuda a criar configurações modulares e mais robustas. Por exemplo, para configurar o logrotate, encontramos o diretório `logrotate.conf`, mas também os diretórios `logrotate.d`.

Esse tratamento é útil nos casos em que aplicativos diferentes precisam de configurações para um mesmo serviço específico. Se, por exemplo, um pacote de servidor web contiver uma configuração de logrotate, essa configuração poderá ser posta em um arquivo dedicado no diretório `logrotate.d`. Esse arquivo pode ser atualizado pelo pacote do servidor web sem interferir na configuração restante de logrotate. Da mesma forma, os pacotes podem adicionar tarefas específicas colocando arquivos no diretório `/etc/cron.d` em vez de modificar `/etc/crontab`.

No Debian - e derivados do Debian - essa abordagem foi aplicada à lista de fontes confiáveis lidas pela ferramenta de gerenciamento de pacotes `apt`: além do clássico `/etc/apt/sources.list`, agora encontramos o diretório `/etc/apt/sources.list.d`:

```
$ ls /etc/apt/sources*
/etc/apt/sources.list
/etc/apt/sources.list.d:
```

Arquivos de configuração em `HOME` (Dotfiles)

No nível do usuário, os programas armazenam suas configurações e arquivos em arquivos ocultos no diretório inicial do usuário (também representado como `~`). Lembre-se de que os arquivos ocultos começam com um ponto (`.`), donde o nome: *dotfiles*.

Alguns desses dotfiles são scripts Bash que personalizam a sessão de shell do usuário, originados assim que o usuário faz login no sistema:

`.bash_history`
: Armazena o histórico da linha de comando.

`.bash_logout`
: Inclui comandos a executar quando o usuário sai do shell de login.

`.bashrc`
: Script de inicialização do Bash para shells não-login.

`.profile`
: Script de inicialização do Bash para shells de login.

NOTE Consulte a lição “As bases da linha de comando” para aprender mais sobre o Bash e seus arquivos de inicialização.

Outros arquivos de configuração específicos ao usuário são obtidos na inicialização dos

programas respectivos: `.gitconfig`, `.emacs.d`, `.ssh` etc.

O kernel do Linux

Antes que qualquer processo possa ser executado, o kernel deve ser carregado em uma área protegida da memória. Depois disso, o processo com o PID `1` (quase sempre o `systemd` hoje em dia) desencadeia a cadeia de processos, ou seja, um processo inicia outro(s) e assim por diante. Uma vez que os processos estão ativos, o kernel do Linux é encarregado de alocar recursos para eles (teclado, mouse, discos, memória, interfaces de rede etc.).

NOTE

Antes do `systemd`, o primeiro processo em um sistema Linux sempre era `/sbin/init`, como parte do gerenciador de sistemas *System V Init*. Na verdade, ainda podemos encontrar o `/sbin/init`, mas vinculado a `/lib/systemd/systemd`.

Onde os Kernels estão armazenados: `/boot`

O kernel reside em `/boot`, junto com outros arquivos relacionados à inicialização. A maioria desses arquivos inclui os componentes do número da versão do kernel em seu nome (versão do kernel, revisão principal, revisão secundária e número do patch).

O diretório `/boot` inclui os seguintes tipos de arquivos, com nomes correspondentes à respectiva versão do kernel:

`config-4.9.0-9-amd64`

Definições de configuração do kernel, como opções e módulos que foram compilados junto com o kernel.

`initrd.img-4.9.0-9-amd64`

Imagem de disco RAM inicial que ajuda no processo de inicialização, carregando um sistema de arquivos root temporário na memória.

`System-map-4.9.0-9-amd64`

O arquivo` System-map` (em certos sistemas, `System.map`) contém localizações de endereços de memória para nomes de símbolos do kernel. Cada vez que um kernel é reconstruído, o conteúdo do arquivo muda, pois as localizações da memória podem ser diferentes. O kernel usa este arquivo para procurar localizações de endereços de memória para um símbolo do kernel determinado, ou vice-versa.

`vmlinuz-4.9.0-9-amd64`

O kernel propriamente dito, em um formato compactado de extração automática (vem daí o `z`

in `vmlinuz`; `vm` significa memória virtual e começou a ser usado quando o kernel começou a ter suporte a memória virtual).

`grub`

Diretório de configuração do gerenciador de inicialização `grub2`.

TIP

Por se tratar de um recurso crítico do sistema operacional, mais de um kernel e seus arquivos associados são mantidos em `/boot`, caso o kernel padrão apresente algum defeito e seja preciso voltar a uma versão anterior para — pelo menos — ser possível inicializar o sistema e repará-lo.

O diretório `/proc`

O diretório `/proc` é um dos chamados sistemas de arquivos virtuais ou pseudo-sistemas de arquivos, pois seu conteúdo não é gravado no disco, mas carregado na memória. Ele é preenchido de forma dinâmica sempre que o computador é inicializado e reflete constantemente o estado atual do sistema. O `/proc` inclui informações sobre:

- Processos em execução
- Configuração do kernel
- Hardware do sistema

Além de todos os dados referentes aos processos que veremos na próxima lição, esse diretório também armazena arquivos com informações sobre o hardware do sistema e as definições de configuração do kernel. Alguns desses arquivos são:

`/proc/cpuinfo`

Armazena informações sobre a CPU do sistema:

```
$ cat /proc/cpuinfo
processor   : 0
vendor_id   : GenuineIntel
cpu family  : 6
model       : 158
model name  : Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz
stepping    : 10
cpu MHz     : 3696.000
cache size  : 12288 KB
(...)
```

`/proc/cmdline`

Armazena as strings passadas para o kernel na inicialização:

```
$ cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-4.9.0-9-amd64 root=UUID=5216e1e4-ae0e-441f-b8f5-8061c0034c74 ro
quiet
```

`/proc/modules`

Mostra a lista de módulos carregados no kernel:

```
$ cat /proc/modules
nls_utf8 16384 1 - Live 0xffffffffc0644000
isofs 40960 1 - Live 0xffffffffc0635000
udf 90112 0 - Live 0xffffffffc061e000
crc_itu_t 16384 1 udf, Live 0xffffffffc04be000
fuse 98304 3 - Live 0xffffffffc0605000
vboxsf 45056 0 - Live 0xffffffffc05f9000 (O)
joydev 20480 0 - Live 0xffffffffc056e000
vboxguest 327680 5 vboxsf, Live 0xffffffffc05a8000 (O)
hid_generic 16384 0 - Live 0xffffffffc0569000
(...)
```

O diretório `/proc/sys`

Este diretório inclui as definições de configuração do kernel em arquivos classificados em categorias por subdiretório:

```
$ ls /proc/sys
abi  debug  dev  fs  kernel  net  user  vm
```

A maioria desses arquivos funciona como um interruptor e — portanto — só contém um dentre dois valores possíveis: `0` ou `1` (“ligado” ou “desligado”). Por exemplo:

`/proc/sys/net/ipv4/ip_forward`

O valor que habilita ou desabilita a máquina para atuar como roteador (capaz de encaminhar pacotes):

```
$ cat /proc/sys/net/ipv4/ip_forward
0
```

Porém, há algumas exceções:

`/proc/sys/kernel/pid_max`

O PID máximo permitido:

```
$ cat /proc/sys/kernel/pid_max
32768
```

WARNING Tenha muito cuidado ao alterar as configurações do kernel, pois o valor errado pode causar instabilidade no sistema.

Dispositivos de hardware

Lembre-se, no Linux "tudo é arquivo". Ou seja, as informações sobre o hardware do dispositivo, bem como as próprias configurações do kernel, são todas armazenadas em arquivos especiais que residem em diretórios virtuais.

O diretório `/dev`

O diretório *device* (dispositivo) `/dev` contém arquivos (ou nós) de dispositivos para todos os dispositivos de hardware conectados. Esses arquivos de dispositivo são usados como interface entre os dispositivos e os processos que os utilizam. Cada arquivo de dispositivo se enquadra em uma destas duas categorias:

Dispositivos de bloco

São aqueles em que os dados são lidos e gravados em blocos que podem ser endereçados individualmente. Podemos dar como exemplo os discos rígidos (e suas partições, como `/dev/sda1`), as unidades flash USB, CDs, DVDs etc.

Dispositivos de caracteres

são aqueles em que os dados são lidos e gravados sequencialmente, um caractere de cada vez, como é o caso dos teclados, console de texto (`/dev/console`), portas seriais (como `/dev/ttyS0` e outras) etc.

Ao listar os arquivos do dispositivo, use `ls` com a opção `-l` para diferenciar os dois. Podemos — por exemplo — verificar discos rígidos e partições:

```
# ls -l /dev/sd*
brw-rw---- 1 root disk 8, 0 may 25 17:02 /dev/sda
brw-rw---- 1 root disk 8, 1 may 25 17:02 /dev/sda1
```

```
brw-rw---- 1 root disk 8, 2 may 25 17:02 /dev/sda2
(...)
```

Ou para terminais seriais (TeleTYpewriter):

```
# ls -l /dev/tty*
crw-rw-rw- 1 root tty     5,  0 may 25 17:26 /dev/tty
crw--w---- 1 root tty     4,  0 may 25 17:26 /dev/tty0
crw--w---- 1 root tty     4,  1 may 25 17:26 /dev/tty1
(...)
```

Observe que o primeiro caractere é `b` para dispositivos de bloco e `c` para dispositivos de caracteres.

TIP

O asterisco (`*`) é um caractere de globbing que significa 0 ou mais caracteres. Por isso, ele é importante nos comandos `ls -l /dev/sd*` e `ls -l /dev/tty*` mostrados acima. Para saber mais sobre esses caracteres especiais, consulte a lição sobre globbing.

Além disso, `/dev` inclui alguns arquivos especiais bastante úteis para diferentes finalidades de programação:

`/dev/zero`
: Fornece tantos caracteres nulos quantos solicitados.

`/dev/null`
: Também chamado de *balde de bits*. Descarta todas as informações enviadas para ele.

`/dev/urandom`
: Gera números pseudo-aleatórios.

O diretório `/sys`

O *sistema de arquivos sys* (`sysfs`) é montado em `/sys`. Foi introduzido com a chegada do kernel 2.6 e representou um grande progresso em `/proc/sys`.

Os processos têm de interagir com os dispositivos em `/dev` e, portanto, o kernel precisa de um diretório que contenha informações sobre esses dispositivos de hardware. Esse diretório é o `/sys`, e seus dados são organizados em categorias. Por exemplo, para verificar o endereço MAC da sua placa de rede (`enp0s3`), aplicaríamos `cat` no seguinte arquivo:

```
$ cat /sys/class/net/enp0s3/address
08:00:27:02:b2:74
```

Memória e tipos de memória

Basicamente, para um programa começar a rodar, ele precisa ser carregado na memória. De modo geral, quando falamos de memória, nos referimos à *Memória de Acesso Randômico* (RAM), que — quando comparada aos discos rígidos mecânicos — tem a vantagem de ser muito mais rápida. Pelo lado negativo, ela é volátil (ou seja, quando o computador é desligado, os dados desaparecem).

Não obstante o mencionado acima, quando se trata de memória, podemos diferenciar dois tipos principais em um sistema Linux:

Memória física

Também conhecida como *RAM*, vem na forma de chips compostos de circuitos integrados contendo milhões de transistores e capacitores. Estes, por sua vez, formam células de memória (o componente básico da memória do computador). Cada uma dessas células tem um código hexadecimal associado — um endereço de memória — que pode ser referenciado quando necessário.

Swap

Também conhecida como *memória de troca*, é a parte da memória virtual que fica no disco rígido e é usada quando não há mais RAM disponível.

Por outro lado, existe o conceito de *memória virtual*, que é uma abstração da quantidade total de memória utilizável, incluindo a memória (a RAM, mas também o espaço em disco) tal como é vista pelos aplicativos.

`free` analisa `/proc/meminfo` e exibe a quantidade de memória livre e usada no sistema de maneira bem clara

```
$ free
               total        used        free      shared  buff/cache   available
Mem:         4050960     1474960     1482260       96900     1093740     2246372
Swap:        4192252           0     4192252
```

Vamos entender melhor as diferentes colunas:

`total`

Quantidade total de memória física e de troca instalada.

`used`

Quantidade de memória física e de troca atualmente em uso.

`free`

Quantidade de memória física e de troca que não está atualmente em uso.

`shared`

Quantidade de memória física usada — principalmente — por `tmpfs`.

`buff/cache`

Quantidade de memória física atualmente em uso pelos buffers do kernel, a cache e as slabs.

`available`

Estimativa da memória física disponível para novos processos.

Por padrão, `free` mostra valores em kibibytes, mas permite uma variedade de opções para exibir os resultados em diferentes unidades de medida. Algumas dessas opções são:

`-b`

Bytes.

`-m`

Mebibytes.

`-g`

Gibibytes.

`-h`

Formato legível por humanos.

`-h` sempre é confortável de ler:

```
$ free -h
               total        used        free      shared  buff/cache   available
Mem:           3,9G        1,4G        1,5G         75M        1,0G        2,2G
Swap:          4,0G          0B        4,0G
```

NOTE Um kibibyte (KiB) é igual a 1.024 bytes, enquanto um kilobyte (KB) é igual a 1000

bytes. O mesmo se aplica, respectivamente, a mebibytes, gibibytes etc.

Exercícios Guiados

1. Use o comando `which` para descobrir a localização dos seguintes programas e preencha a tabela:

Programa	**comando `which`**	**Caminho para o executável (saída)**	**O usuário precisa de privilégios de `root`?**
`swapon`			
`kill`			
`cut`			
`usermod`			
`cron`			
`ps`			

2. Onde se encontram os arquivos a seguir?

Arquivo	`/etc`	`~`
`.bashrc`		
`bash.bashrc`		
`passwd`		
`.profile`		
`resolv.conf`		
`sysctl.conf`		

3. Explique o significado dos elementos numéricos no arquivo do kernel `vmlinuz-4.15.0-50-generic` encontrado em `/boot`:

Elemento numérico	**Significado**
`4`	
`15`	
`0`	
`50`	

4. Qual comando você usaria para listar todos os discos rígidos e partições em `/dev`?

Exercícios Exploratórios

1. Os arquivos de dispositivo para discos rígidos são representados com base nos controladores que eles usam — vimos `/dev/sd*` para unidades que usam SCSI (Small Computer System Interface) e SATA (Serial Advanced Technology Attachment), mas
   - Como as antigas unidades IDE (Integrated Drive Electronics) eram representadas?
   - E as unidades NVMe (Non-Volatile Memory Express) modernas?

2. Analise o arquivo `/proc/meminfo`. Compare o conteúdo desse arquivo com a saída do comando `free` e identifique qual chave de `/proc/meminfo` corresponde aos campos seguintes na saída de `free`:

saída de `free`	**campo** `/proc/meminfo`
`total`	
`free`	
`shared`	
`buff/cache`	
`available`	

Resumo

Nesta lição, você aprendeu sobre a localização dos programas e seus arquivos de configuração em um sistema Linux. O que é importante lembrar:

- Basicamente, os programas são encontrados em uma estrutura de diretórios em três níveis: `/`, `/usr` e `/usr/local`. Cada um desses níveis pode conter diretórios `bin` e `sbin`.
- Os arquivos de configuração são armazenados em `/etc` e `~`.
- Os dotfiles são arquivos ocultos que começam com um ponto (`.`).

Também falamos sobre o kernel do Linux. Conceitos importantes:

- Para o Linux, tudo é arquivo.
- O kernel do Linux mora em `/boot`, junto com outros arquivos relacionados à inicialização.
- Para os processos começarem a rodar, o kernel precisa ser carregado primeiro em uma área protegida da memória.
- O trabalho do kernel é alocar recursos do sistema aos processos.
- O sistema de arquivos virtual (ou pseudo) `/proc` armazena dados importantes do kernel e do sistema de forma volátil.

Da mesma forma, exploramos os dispositivos de hardware e aprendemos o seguinte:

- O diretório `/dev` armazena arquivos especiais (também chamados de nós) para todos os dispositivos de hardware conectados: *dispositivos de bloco* ou *dispositivos de caracteres*. Os primeiros transferem os dados em blocos; os outros, um caractere de cada vez.
- O diretório `/dev` também contém outros arquivos especiais como `/dev/zero`, `/dev/null` ou `/dev/urandom`.
- O diretório `/sys` armazena informações sobre dispositivos de hardware, organizadas em categorias.

Finalmente, falamos um pouco da memória. Nós aprendemos que:

- Um programa roda quando é carregado na memória.
- O que é RAM (Memória de Acesso Randômico).
- O que é Swap ou troca.
- Como exibir o uso da memória.

Comandos usados nesta lição:

`cat`

Concatena/imprime o conteúdo do arquivo.

`free`

Exibe a quantidade de memória livre e usada no sistema.

`ls`

Lista o conteúdo do diretório.

`which`

Mostra a localização do programa.

Respostas aos Exercícios Guiados

1. Use o comando `which` para descobrir a localização dos seguintes programas e preencha a tabela:

Programa	**comando `which`**	**Caminho para o executável (saída)**	**O usuário precisa de privilégios de `root`?**
`swapon`	`which swapon`	`/sbin/swapon`	Sim
`kill`	`which kill`	`/bin/kill`	Não
`cut`	`which cut`	`/usr/bin/cut`	Não
`usermod`	`which usermod`	`/usr/sbin/usermod`	Sim
`cron`	`which cron`	`/usr/sbin/cron`	Sim
`ps`	`which ps`	`/bin/ps`	Não

2. Onde se encontram os arquivos a seguir?

Arquivo	`/etc`	`~`
`.bashrc`	Não	Sim
`bash.bashrc`	Sim	Não
`passwd`	Sim	Não
`.profile`	Não	Sim
`resolv.conf`	Sim	Não
`sysctl.conf`	Sim	Não

3. Explique o significado dos elementos numéricos no arquivo do kernel `vmlinuz-4.15.0-50-generic` encontrado em `/boot`:

Elemento numérico	**Significado**
`4`	Versão do kernel
`15`	Revisão importante
`0`	Revisão secundária
`50`	Número do patch

4. Qual comando você usaria para listar todos os discos rígidos e partições em `/dev`?

```
ls /dev/sd*
```

Respostas aos Exercícios Exploratórios

1. Os arquivos de dispositivo para discos rígidos são representados com base nos controladores que eles usam — vimos `/dev/sd*` para unidades que usam SCSI (Small Computer System Interface) e SATA (Serial Advanced Technology Attachment), mas

   `/dev/hd*`

   - E as unidades NVMe (Non-Volatile Memory Express) modernas?

     `/dev/nvme*`

2. Analise o arquivo `/proc/meminfo`. Compare o conteúdo desse arquivo com a saída do comando `free` e identifique qual chave de `/proc/meminfo` corresponde aos campos seguintes na saída de `free`:

saída de `free`	**campo** `/proc/meminfo`
`total`	`MemTotal` / `SwapTotal`
`free`	`MemFree` / `SwapFree`
`shared`	`Shmem`
`buff/cache`	`Buffers`, `Cached` and `SReclaimable`
`available`	`MemAvailable`

4.3 Lição 2

Certificação:	Linux Essentials
Versão:	1.6
Tópico:	4 O sistema operacional Linux
Objetivo:	4.3 Onde os dados são armazenados
Lição:	2 de 2

Introdução

Depois de explorar os programas e seus arquivos de configuração, vamos aprender como os comandos são executados como processos. Da mesma forma, trataremos das mensagens do sistema, do uso do buffer de anel do kernel e de como a chegada do `systemd` e seu daemon de diário — `journald` — mudou a maneira como as coisas eram feitas até então com relação aos registros do sistema.

Processos

Sempre que um usuário emite um comando, um programa é executado e um ou mais processos são gerados.

Os processos existem dentro de uma hierarquia. Depois que o kernel é carregado na memória durante a inicialização, é iniciado o primeiro processo que — por sua vez — inicia outros processos, os quais também podem iniciar outros processos. Cada processo possui um identificador exclusivo (`PID`) e um identificador do processo pai (`PPID`). Estes são números inteiros positivos atribuídos em ordem sequencial.

Explorando os processos dinamicamente: `top`

Para obter uma lista dinâmica de todos os processos em execução, usamos o comando `top`:

```
$ top

top - 11:10:29 up  2:21,  1 user,  load average: 0,11, 0,20, 0,14
Tasks:  73 total,   1 running,  72 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0,0 us,  0,3 sy,  0,0 ni, 99,7 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
KiB Mem :  1020332 total,   909492 free,    38796 used,    72044 buff/cache
KiB Swap:  1046524 total,  1046524 free,        0 used.   873264 avail Mem

   PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
   436 carol     20   0   42696   3624   3060 R  0,7  0,4   0:00.30 top
     4 root      20   0       0      0      0 S  0,3  0,0   0:00.12 kworker/0:0
   399 root      20   0   95204   6748   5780 S  0,3  0,7   0:00.22 sshd
     1 root      20   0   56872   6596   5208 S  0,0  0,6   0:01.29 systemd
     2 root      20   0       0      0      0 S  0,0  0,0   0:00.00 kthreadd
     3 root      20   0       0      0      0 S  0,0  0,0   0:00.02 ksoftirqd/0
     5 root       0 -20       0      0      0 S  0,0  0,0   0:00.00 kworker/0:0H
     6 root      20   0       0      0      0 S  0,0  0,0   0:00.00 kworker/u2:0
     7 root      20   0       0      0      0 S  0,0  0,0   0:00.08 rcu_sched
     8 root      20   0       0      0      0 S  0,0  0,0   0:00.00 rcu_bh
     9 root      rt   0       0      0      0 S  0,0  0,0   0:00.00 migration/0
    10 root       0 -20       0      0      0 S  0,0  0,0   0:00.00 lru-add-drain
    (...)
```

Como vimos acima, `top` também fornece informações sobre o consumo de memória e CPU do sistema em geral, bem como sobre cada processo.

O `top` permite uma certa interação do usuário.

Por padrão, a saída é classificada pela porcentagem de tempo da CPU usada por cada processo em ordem decrescente. Esse comportamento pode ser modificado pressionando as seguintes teclas de dentro de `top`:

`M`

Classifica por uso da memória.

`N`

Classifica pelo número de identificação do processo.

T

Classifica pelo tempo de execução.

P

Classifica pela porcentagem de uso da CPU.

Para alternar entre ordem crescente e decrescente, pressione `R`.

TIP Uma versão mais sofisticada e amigável do `top` é `htop`. Outra alternativa — talvez mais exaustiva — é `atop`. Se eles ainda não estiverem instalados em seu sistema, use seu gerenciador de pacotes para instalá-los e experimentá-los.

Um instantâneo dos processos: `ps`

Outro comando muito útil para obter informações sobre processos é o `ps`. Enquanto `top` fornece informações dinâmicas, as do `ps` são estáticas.

Se invocado sem opções, a saída do `ps` é bastante específica e se refere apenas aos processos vinculados ao shell atual:

```
$ ps
  PID TTY          TIME CMD
 2318 pts/0    00:00:00 bash
 2443 pts/0    00:00:00 ps
```

As informações exibidas relacionam-se ao identificador do processo (`PID`), ao terminal no qual o processo está sendo executado (`TTY`), ao tempo de CPU usado pelo processo (`TIME`) e ao comando que iniciou o processo (`CMD`).

Uma opção útil para o `ps` é `-f`, que mostra a listagem em formato completo:

```
$ ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
carol     2318  1682  0 08:38 pts/1    00:00:00 bash
carol     2443  2318  0 08:46 pts/1    00:00:00 ps -f
```

Em combinação com outras opções, `-f` mostra a relação entre os processos pai e filho:

```
$ ps -uf
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
```

```
carol       2318  0.0  0.1  21336  5140 pts/1    Ss   08:38   0:00 bash
carol       2492  0.0  0.0  38304  3332 pts/1    R+   08:51   0:00  \_ ps -uf
carol       1780  0.0  0.1  21440  5412 pts/0    Ss   08:28   0:00 bash
carol       2291  0.0  0.7 305352 28736 pts/0    Sl+  08:35   0:00  \_ emacs index.en.adoc
-nw
(...)
```

Da mesma forma, o `ps` pode exibir a porcentagem de memória usada quando invocado com a opção `-v`:

```
$ ps -v
  PID TTY      STAT   TIME  MAJFL   TRS   DRS   RSS %MEM COMMAND
 1163 tty2     Ssl+   0:00      1    67 201224 5576  0.1 /usr/lib/gdm3/gdm-x-session (...)
 (...)
```

NOTE Um outro comando visualmente atraente para exibir a hierarquia de processos é o `pstree`. Ele está incluído em todas as principais distribuições.

Informações sobre processos no diretório `/proc`

Já vimos o sistema de arquivos `/proc`. O `/proc` inclui um subdiretório numerado para cada processo em execução no sistema (o número é o `PID` do processo):

```
carol@debian:~# ls /proc
1    108  13   17   21   27   354  41   665  8    9
10   109  14   173  22   28   355  42   7    804  915
103  11   140  18   23   29   356  428  749  810  918
104  111  148  181  24   3    367  432  75   811
105  112  149  19   244  349  370  433  768  83
106  115  15   195  25   350  371  5    797  838
107  12   16   2    26   353  404  507  798  899
(...)
```

Assim, todas as informações sobre um processo específico são incluídas dentro de seu diretório. Vamos listar o conteúdo do primeiro processo — aquele cujo `PID` é `1` (a saída foi truncada para facilitar a leitura):

```
# ls /proc/1/
attr        cmdline          environ  io        mem       ns
autogroup   comm             exe      limits    mountinfo numa_maps
```

```
auxv        coredump_filter  fd       loginuid  mounts    oom_adj
...
```

Você pode verificar — por exemplo — o executável do processo:

```
# cat /proc/1/cmdline; echo
/sbin/init
```

Como vemos, o binário que iniciou a hierarquia de processos era `/sbin/init`.

NOTE Podemos concatenar comandos com ponto e vírgula (`;`). A razão para usar o comando `echo` acima é fornecer uma nova linha. Experimente executar simplesmente `cat /proc/1/cmdline` para ver a diferença.

A carga do sistema

Potencialmente, cada processo em um sistema consome recursos. A chamada carga do sistema tenta agregar a carga geral do sistema em um único indicador numérico. Podemos ver a carga atual com o comando `uptime`:

```
$ uptime
 22:12:54 up 13 days, 20:26,  1 user,  load average: 2.91, 1.59, 0.39
```

Os três últimos dígitos indicam a média de carga do sistema para o último minuto (`2,91`), os últimos cinco minutos (`1.59`) e os últimos quinze minutos (`0.39`), respectivamente.

Cada um desses números indica quantos processos estavam aguardando recursos da CPU ou a conclusão das operações de entrada/saída. Isso significa que esses processos estavam prontos para a execução quando recebessem os respectivos recursos.

Registro do sistema e mensagens do sistema

Quando o kernel e os processos começam ser executados e a se comunicar entre si, muitas informações são produzidas. A maior parte é enviada para arquivos — os chamados arquivos de registro ou, simplesmente, *logs*.

Sem esses registros, a busca por um evento que ocorreu em um servidor seria uma verdadeira dor de cabeça para os administradores de sistemas, daí a importância de ter uma maneira padronizada e centralizada de acompanhar os eventos do sistema. Além disso, os logs são claros e determinantes para a solução de problemas e questões de segurança, além de serem fontes de

dados confiáveis para entender as estatísticas do sistema e prever tendências.

Registros com o Daemon syslog

Tradicionalmente, as mensagens do sistema são gerenciadas pelo recurso de registro padrão — syslog — ou um de seus derivados — syslog-ng ou rsyslog. O daemon de log coleta mensagens de outros serviços e programas e as armazena em arquivos de log, normalmente em `/var/log`. No entanto, alguns serviços cuidam de seus próprios logs (por exemplo, o servidor web Apache HTTPD). Da mesma forma, o kernel do Linux usa um buffer de anel na memória para armazenar suas mensagens de log.

Arquivos de registro em `/var/log`

Como os registros são dados que variam ao longo do tempo, normalmente eles são encontrados em `/var/log`.

Se você explorar `/var/log`, perceberá que os nomes dos logs são — até certo ponto — bastante auto-explicativos. Eis alguns exemplos:

`/var/log/auth.log`
: Armazena informações sobre a autenticação.

`/var/log/kern.log`
: Armazena informações do kernel.

`/var/log/syslog`
: Armazena informações do sistema.

`/var/log/messages`
: Armazena dados do sistema e aplicativos.

NOTE O nome exato e o conteúdo dos arquivos de log podem variar nas diferentes distribuições Linux.

Acessando arquivos de log

Ao explorar arquivos de log, você precisa ser root (se não tiver permissões de leitura) e usar um paginador como `less`:

```
# less /var/log/messages
Jun  4 18:22:48 debian liblogging-stdlog:  [origin software="rsyslogd" swVersion="8.24.0" x-
```

```
pid="285" x-info="http://www.rsyslog.com"] rsyslogd was HUPed
Jun 29 16:57:10 debian kernel: [    0.000000] Linux version 4.9.0-8-amd64 (debian-
kernel@lists.debian.org) (gcc version 6.3.0 20170516 (Debian 6.3.0-18+deb9u1) ) #1 SMP
Debian 4.9.130-2 (2018-10-27)
Jun 29 16:57:10 debian kernel: [    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-4.9.0-
8-amd64 root=/dev/sda1 ro quiet
```

Outra alternativa é usar `tail` com a opção `-f` para ler as mensagens mais recentes do arquivo e exibir dinamicamente as novas linhas conforme elas são incluídas:

```
# tail -f /var/log/messages
Jul  9 18:39:37 debian kernel: [    2.350572] RAPL PMU: hw unit of domain psys 2^-0 Joules
Jul  9 18:39:37 debian kernel: [    2.512802] input: VirtualBox USB Tablet as
/devices/pci0000:00/0000:00:06.0/usb1/1-1/1-1:1.0/0003:80EE:0021.0001/input/input7
Jul  9 18:39:37 debian kernel: [    2.513861] Adding 1046524k swap on /dev/sda5.  Priority:-
1 extents:1 across:1046524k FS
Jul  9 18:39:37 debian kernel: [    2.519301] hid-generic 0003:80EE:0021.0001:
input,hidraw0: USB HID v1.10 Mouse [VirtualBox USB Tablet] on usb-0000:00:06.0-1/input0
Jul  9 18:39:37 debian kernel: [    2.623947] snd_intel8x0 0000:00:05.0: white list rate for
1028:0177 is 48000
Jul  9 18:39:37 debian kernel: [    2.914805] IPv6: ADDRCONF(NETDEV_UP): enp0s3: link is not
ready
Jul  9 18:39:39 debian kernel: [    4.937283] e1000: enp0s3 NIC Link is Up 1000 Mbps Full
Duplex, Flow Control: RX
Jul  9 18:39:39 debian kernel: [    4.938493] IPv6: ADDRCONF(NETDEV_CHANGE): enp0s3: link
becomes ready
Jul  9 18:39:40 debian kernel: [    5.315603] random: crng init done
Jul  9 18:39:40 debian kernel: [    5.315608] random: 7 urandom warning(s) missed due to
ratelimiting
```

A saída aparecerá no seguinte formato:

- Registro de data e hora
- Nome do host do qual a mensagem veio
- Nome do programa/serviço que gerou a mensagem
- PID do programa que gerou a mensagem
- Descrição da ação que ocorreu

A maioria dos arquivos de registro é gravada em texto simples; no entanto, alguns podem conter dados binários, como é o caso de `/var/log/wtmp` — que armazena dados relevantes para logins

bem-sucedidos. Você pode usar o comando `file` para determinar qual é o caso:

```
$ file /var/log/wtmp
/var/log/wtmp: dBase III DBT, version number 0, next free block index 8
```

Esses arquivos são normalmente lidos usando comandos especiais. `last` é usado para interpretar os dados em `/var/log/wtmp`:

```
$ last
carol    tty2         :0               Thu May 30 10:53   still logged in
reboot   system boot  4.9.0-9-amd64    Thu May 30 10:52   still running
carol    tty2         :0               Thu May 30 10:47 - crash  (00:05)
reboot   system boot  4.9.0-9-amd64    Thu May 30 09:11   still running
carol    tty2         :0               Tue May 28 08:28 - 14:11  (05:42)
reboot   system boot  4.9.0-9-amd64    Tue May 28 08:27 - 14:11  (05:43)
carol    tty2         :0               Mon May 27 19:40 - 19:52  (00:11)
reboot   system boot  4.9.0-9-amd64    Mon May 27 19:38 - 19:52  (00:13)
carol    tty2         :0               Mon May 27 19:35 - down   (00:03)
reboot   system boot  4.9.0-9-amd64    Mon May 27 19:34 - 19:38  (00:04)
```

NOTE Assim como `/var/log/wtmp`, `/var/log/btmp` armazena informações sobre tentativas malsucedidas de login. O comando especial para ler seu conteúdo é `lastb`.

Rotação do log

Os arquivos de log podem crescer muito em algumas semanas ou meses e ocupar todo o espaço livre em disco. Para resolver isso, usamos o utilitário `logrotate`. Ele implementa a rotação ou ciclo de log, que implica ações como mover arquivos de log para um novo nome, arquivando-os e/ou compactando-os, às vezes enviando-os por email para o administrador do sistema e, eventualmente, excluindo-os à medida que envelhecem. Há várias convenções usadas para nomear esses arquivos de log rotacionados (por exemplo, adicionando um sufixo com a data); no entanto, o mais comum é adicionar um sufixo com um número inteiro:

```
# ls /var/log/apache2/
access.log  error.log  error.log.1  error.log.2.gz  other_vhosts_access.log
```

Note que `error.log.2.gz` já foi compactado com o `gzip` (daí o sufixo `.gz`).

O buffer do anel do kernel

O buffer de anel do kernel é uma estrutura de dados de tamanho fixo que registra as mensagens de inicialização do kernel. A função desse buffer — importantíssima — é registrar todas as mensagens do kernel produzidas na inicialização — quando o `syslog` ainda não está disponível. O comando `dmesg` imprime na tela o buffer de anel do kernel (que também costumava ser armazenado em `/var/log/dmesg`). Devido à extensão do buffer de anel, este comando é normalmente usado em combinação com o utilitário de filtragem de texto `grep` ou um paginador como o `less`. Por exemplo, para procurar mensagens de inicialização:

```
$ dmesg | grep boot
[    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-4.9.0-9-amd64 root=UUID=5216e1e4-ae0e-
441f-b8f5-8061c0034c74 ro quiet
[    0.000000] smpboot: Allowing 1 CPUs, 0 hotplug CPUs
[    0.000000] Kernel command line: BOOT_IMAGE=/boot/vmlinuz-4.9.0-9-amd64
root=UUID=5216e1e4-ae0e-441f-b8f5-8061c0034c74 ro quiet
[    0.144986] AppArmor: AppArmor disabled by boot time parameter
(...)
```

NOTE Conforme o buffer de anel do kernel vai crescendo com novas mensagens, as mais antigas desaparecem.

O diário do sistema: systemd-journald

A partir de 2015, o systemd substituiu o SysV Init como gerenciador de sistemas e serviços na maioria das principais distribuições Linux. Como conseqüência, o daemon do diário — journald — tornou-se o componente de registro padrão, sobrepondo-se ao syslog na maioria dos aspectos. Os dados não são mais armazenados em texto simples, mas em formato binário. Assim, o utilitário `journalctl` é necessário para ler os logs. Além disso, o journald é compatível com syslog e pode ser integrado ao syslog.

`journalctl` é o utilitário usado para ler e consultar o banco de dados do diário do systemd. Quando invocado sem opções, ele imprime o diário inteiro:

```
# journalctl
-- Logs begin at Tue 2019-06-04 17:49:40 CEST, end at Tue 2019-06-04 18:13:10 CEST. --
jun 04 17:49:40 debian systemd-journald[339]: Runtime journal (/run/log/journal/) is 8.0M,
max 159.6M, 151.6M free.
jun 04 17:49:40 debian kernel: microcode: microcode updated early to revision 0xcc, date =
2019-04-01
Jun 04 17:49:40 debian kernel: Linux version 4.9.0-8-amd64 (debian-kernel@lists.debian.org)
```

```
(gcc version 6.3.0 20170516 (Debian 6.3.0-18+deb9u1) )
Jun 04 17:49:40 debian kernel: Command line: BOOT_IMAGE=/boot/vmlinuz-4.9.0-8-amd64
root=/dev/sda1 ro quiet
(...)
```

No entanto, se chamado com as opções `-k` ou `--dmesg`, será equivalente ao uso do comando `dmesg`:

```
# journalctl -k
[    0.000000] Linux version 4.9.0-9-amd64 (debian-kernel@lists.debian.org) (gcc version
6.3.0 20170516 (Debian 6.3.0-18+deb9u1) ) #1 SMP Debian 4.9.168-1+deb9u2 (2019-05-13)
[    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-4.9.0-9-amd64 root=UUID=5216e1e4-ae0e-
441f-b8f5-8061c0034c74 ro quiet
(...)
```

Dentre as outras opções interessantes para o `journalctl` estão:

`-b`, **`--boot`**

Mostra informações de inicialização.

`-u`

Mostra mensagens sobre uma unidade especificada. Grosso modo, uma unidade pode ser definida como qualquer recurso tratado pelo systemd. Por exemplo, `journalctl -u apache2.service` é usado para ler mensagens sobre o servidor web `apache2`.

`-f`

Mostra as mensagens mais recentes do diário e segue imprimindo novas entradas à medida que elas são incluídas no diário — a exemplo de `tail -f`.

Exercícios Guiados

1. Analise esta lista de `top` e responda às questões a seguir:

```
carol@debian:~$ top

top - 13:39:16 up 31 min,  1 user,  load average: 0.12, 0.15, 0.10
Tasks:  73 total,   2 running,  71 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.1 us,  0.4 sy,  0.0 ni, 98.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  1020332 total,   698700 free,   170664 used,   150968 buff/cache
KiB Swap:  1046524 total,  1046524 free,        0 used.   710956 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
  605 nobody    20   0 1137620 132424  34256 S  6.3 13.0   1:47.24 ntopng
  444 www-data  20   0  364780   4132   2572 S  0.3  0.4   0:00.44 apache2
  734 root      20   0   95212   7004   6036 S  0.3  0.7   0:00.36 sshd
  887 carol     20   0   46608   3680   3104 R  0.3  0.4   0:00.03 top
    1 root      20   0   56988   6688   5240 S  0.0  0.7   0:00.42 systemd
    2 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kthreadd
    3 root      20   0       0      0      0 S  0.0  0.0   0:00.09 ksoftirqd/0
    4 root      20   0       0      0      0 S  0.0  0.0   0:00.87 kworker/0:0
(...)
```

   - Quais processos foram iniciados pelo usuário `carol`?

   - Qual diretório virtual de `/proc` você deve visitar para encontrar dados relacionados ao comando `top`?

   - Qual processo foi executado primeiro? Como você descobriu?

   - Complete a tabela especificando em qual área da saída de `top` encontramos as seguintes informações:

Informações sobre ...	**Área de resumo**	**Área de tarefas**
Memória		
Swap		
PID		

Informações sobre ...	Área de resumo	Área de tarefas
Tempo da CPU		
Comandos		

2. Qual comando é usado para ler os registros binários a seguir?
   - `/var/log/wtmp`
   - `/var/log/btmp`
   - `/run/log/journal/2a7d9730cd3142f4b15e20d6be631836/system.journal`
3. Em combinação com `grep`, quais comandos você usaria para descobrir as seguintes informações sobre seu sistema Linux?
   - Data da última reinicialização do sistema (`wtmp`)
   - Quais são os discos rígidos instalados (`kern.log`)
   - Quando foi feito o último login (`auth.log`)
4. Quais dois comandos você usaria para exibir o buffer de anel do kernel?
5. Indique o local a que pertencem as seguintes mensagens de log:
   - `Jul 10 13:37:39 debian dbus[303]: [system] Successfully activated service 'org.freedesktop.nm_dispatcher'`

`/var/log/auth.log`	
`/var/log/kern.log`	
`/var/log/syslog`	
`/var/log/messages`	

- `Jul 10 11:23:58 debian kernel: [ 1.923349] usbhid: USB HID core driver`

`/var/log/auth.log`	
`/var/log/kern.log`	
`/var/log/syslog`	
`/var/log/messages`	

- `Jul 10 14:02:53 debian sudo: pam_unix(sudo:session): session opened for user root by carol(uid=0)`

`/var/log/auth.log`	
`/var/log/kern.log`	
`/var/log/syslog`	
`/var/log/messages`	

- `Jul 10 11:23:58 debian NetworkManager[322]: <info> [1562750638.8672] NetworkManager (version 1.6.2) is starting…`

`/var/log/auth.log`	
`/var/log/kern.log`	
`/var/log/syslog`	
`/var/log/messages`	

6. Use `journalctl` para encontrar informações sobre as seguintes unidades:

Unidade	**Comando**
`ssh`	
`networking`	
`rsyslog`	
`cron`	

Exercícios Exploratórios

1. Retome a saída de `top` dos exercícios guiados e responda às seguintes questões:
   - Quais são as duas etapas a seguir para eliminar o servidor web *apache*?
   - Na área de resumo, como seria possivel exibir as informações sobre a memória física e a memória swap usando barras de progresso?
   - Agora, classifique os processos pelo uso da memória:
   - Agora que as informações sobre a memória estão sendo exibidas em barras de progresso e os processos classificados pelo uso da memória, salve essas configurações para torná-las padrão na próxima vez que você usar o `top`:
   - Qual arquivo armazena as configurações de `top`? Onde ele fica? Como você pode verificar sua existência?
2. Aprenda sobre o comando `exec` no Bash. Tente demonstrar sua funcionalidade iniciando uma sessão do Bash e encontrando o processo do Bash com `ps`. Depois, execute `exec /bin/sh` e busque novamente pelo processo com o mesmo PID.
3. Siga estes passos para explorar os eventos do kernel e o gerenciamento dinâmico de dispositivos do udev:
   - Conecte um drive USB ao seu computador. Rode o `dmesg` e preste atenção às últimas linhas. Qual a linha mais recente?
   - Tendo em mente a saída do comando anterior, execute `ls /dev/sd*` e verifique se seu drive USB aparece na lista. Qual a saída?
   - Agora, remova o drive USB e execute `dmesg` outra vez. O que diz a linha mais recente?
   - Execute `ls /dev/sd*` novamente e verifique se o dispositivo sumiu da lista. Qual a saída?

Resumo

No contexto do armazenamento de dados, os seguintes tópicos foram discutidos nesta lição: gerenciamento de processos e registros e mensagens do sistema.

Em relação ao gerenciamento de processos, aprendemos o seguinte:

- Os programas geram processos e os processos existem numa hierarquia.
- Cada processo tem um identificador único (`PID`) e um identificador do processo pai (`PPID`).
- `top` é um comando muito útil para explorar de maneira dinâmica e interativa os processos em execução no sistema.
- `ps` pode ser usado para obter um instantâneo dos processos atualmente em execução no sistema.
- O diretório `/proc` inclui diretórios para cada um dos processos em execução no sistema, nomeados de acordo com o PID de cada um.
- O conceito de carga média do sistema — utilíssimo para conferir a utilização/sobrecarga da CPU.

Em relação ao log do sistema, devemos lembrar que:

- Um log é um arquivo em que os eventos do sistema são registrados. Os logs são inestimáveis para a solução de problemas.
- O log tradicionalmente é tratado por serviços especiais como syslog, syslog-ng ou rsyslog. No entanto, alguns programas usam seus próprios daemons de registro.
- Como os logs são dados variáveis, eles são mantidos em `/var` e — às vezes — seus nomes dão uma pista sobre o conteúdo (`kern.log`, `auth.log` etc.).
- A maioria dos logs é escrita em texto simples e pode ser lida com qualquer editor de texto, desde que você tenha as permissões corretas. No entanto, alguns deles são binários e devem ser lidos usando comandos especiais.
- Para evitar problemas com espaço em disco, uma *rotação do log* é realizada pelo utilitário logrotate.
- Quanto ao kernel, ele usa uma estrutura circular de dados — o buffer de anel — na qual as mensagens de inicialização são mantidas (as mensagens antigas desaparecem com o tempo).
- O gerenciador de sistema e serviços systemd substituiu o init do System V em praticamente todas as distros, tendo o journald se tornado o serviço de registro padrão.
- Para ler o diário do systemd, é necessário o utilitário `journalctl`.

Comandos usados nesta lição:

`cat`
: Concatena/imprime o conteúdo do arquivo.

`dmesg`
: Imprime o buffer do anel do kernel.

`echo`
: Exibe uma linha de texto ou uma nova linha.

`file`
: Determina o tipo de arquivo.

`grep`
: Imprime linhas de acordo com um padrão.

`last`
: Mostra uma lista dos últimos usuários conectados.

`less`
: Exibe o conteúdo do arquivo uma página de cada vez.

`ls`
: Lista o conteúdo do diretório.

`journalctl`
: Consulta o diário `systemd`.

`tail`
: Exibe as últimas linhas de um arquivo.

Respostas aos Exercícios Guiados

1. Analise esta lista de `top` e responda às questões a seguir:

```
carol@debian:~$ top

top - 13:39:16 up 31 min,  1 user,  load average: 0.12, 0.15, 0.10
Tasks:  73 total,   2 running,  71 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.1 us,  0.4 sy,  0.0 ni, 98.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  1020332 total,   698700 free,   170664 used,   150968 buff/cache
KiB Swap:  1046524 total,  1046524 free,        0 used.   710956 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
  605 nobody    20   0 1137620 132424  34256 S  6.3 13.0   1:47.24 ntopng
  444 www-data  20   0  364780   4132   2572 S  0.3  0.4   0:00.44 apache2
  734 root      20   0   95212   7004   6036 S  0.3  0.7   0:00.36 sshd
  887 carol     20   0   46608   3680   3104 R  0.3  0.4   0:00.03 top
    1 root      20   0   56988   6688   5240 S  0.0  0.7   0:00.42 systemd
    2 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kthreadd
    3 root      20   0       0      0      0 S  0.0  0.0   0:00.09 ksoftirqd/0
    4 root      20   0       0      0      0 S  0.0  0.0   0:00.87 kworker/0:0
(...)
```

   - Quais processos foram iniciados pelo usuário `carol`?

     Resposta: apenas um: `top`.

   - Qual diretório virtual de `/proc` você deve visitar para encontrar dados relacionados ao comando `top`?

     Resposta: `/proc/887`

   - Qual processo foi executado primeiro? Como você descobriu?

     Resposta: `systemd`. Porque ele tem o `PID` #1.

   - Complete a tabela especificando em qual área da saída de `top` encontramos as seguintes informações:

Informações sobre ...	**Área de resumo**	**Área de tarefas**
Memória	Sim	Sim

Informações sobre ...	Área de resumo	Área de tarefas
Swap	Sim	Não
`PID`	Não	Sim
Tempo da CPU	Sim	Sim
Comandos	Não	Sim

2. Qual comando é usado para ler os registros binários a seguir?
   - `/var/log/wtmp`

     Resposta: `last`
   - `/var/log/btmp`

     Resposta: `lastb`
   - `/run/log/journal/2a7d9730cd3142f4b15e20d6be631836/system.journal`

     Resposta: `journalctl`
3. Em combinação com `grep`, quais comandos você usaria para descobrir as seguintes informações sobre seu sistema Linux?
   - Data da última reinicialização do sistema (`wtmp`)

     Resposta: `last`
   - Quais são os discos rígidos instalados (`kern.log`)

     Resposta: `less /var/log/kern.log`
   - Quando foi feito o último login (`auth.log`)

     Resposta: `less /var/log/auth.log`
4. Quais dois comandos você usaria para exibir o buffer de anel do kernel?

   `dmesg` e `journalctl -k` (e também `journalctl --dmesg`).
5. Indique o local a que pertencem as seguintes mensagens de log:
   - `Jul 10 13:37:39 debian dbus[303]: [system] Successfully activated service 'org.freedesktop.nm_dispatcher'`

`/var/log/auth.log`	
`/var/log/kern.log`	
`/var/log/syslog`	X
`/var/log/messages`	

- `Jul 10 11:23:58 debian kernel: [ 1.923349] usbhid: USB HID core driver`

`/var/log/auth.log`	
`/var/log/kern.log`	X
`/var/log/syslog`	
`/var/log/messages`	X

- `Jul 10 14:02:53 debian sudo: pam_unix(sudo:session): session opened for user root by carol(uid=0)`

`/var/log/auth.log`	X
`/var/log/kern.log`	
`/var/log/syslog`	
`/var/log/messages`	

- `Jul 10 11:23:58 debian NetworkManager[322]: <info> [1562750638.8672] NetworkManager (version 1.6.2) is starting…`

`/var/log/auth.log`	
`/var/log/kern.log`	
`/var/log/syslog`	
`/var/log/messages`	X

6. Use `journalctl` para encontrar informações sobre as seguintes unidades:

Unidade	**Comando**
`ssh`	`journalctl -u ssh.service`
`networking`	`journalctl -u networking.service`
`rsyslog`	`journalctl -u rsyslog.service`

Unidade	Comando
`cron`	`journalctl -u cron.service`

Respostas aos Exercícios Exploratórios

1. Retome a saída de `top` dos exercícios guiados e responda às seguintes questões:
   - Quais são as duas etapas a seguir para eliminar o servidor web *apache*?

     Primeiro, pressionar `k`; depois fornecer um valor `kill`.

   - Na área de resumo, como seria possivel exibir as informações sobre a memória física e a memória swap usando barras de progresso?

     Pressionando `m` uma ou duas vezes.

   - Agora, classifique os processos pelo uso da memória:

     `M`

   - Agora que as informações sobre a memória estão sendo exibidas em barras de progresso e os processos classificados pelo uso da memória, salve essas configurações para torná-las padrão na próxima vez que você usar o `top`:

     `W`

   - Qual arquivo armazena as configurações de `top`? Onde ele fica? Como você pode verificar sua existência?

     O arquivo é `~/.config/procps/toprc` e ele fica no diretório inicial do usuário (`~`). Como se trata de um arquivo oculto (reside em um diretório cujo nome começa com um ponto), podemos verificar sua existência com `ls -a` (listar todos os arquivos). Esse arquivo pode ser gerado pressionando `Shift` + `W` enquanto estamos em `top`.

2. Aprenda sobre o comando `exec` no Bash. Tente demonstrar sua funcionalidade iniciando uma sessão do Bash e encontrando o processo do Bash com `ps`. Depois, execute `exec /bin/sh` e busque novamente pelo processo com o mesmo PID.

   `exec` substitui um processo por outro comando. No exemplo a seguir vemos que o processo do Bash é substituído por `/bin/sh` (em vez de `/bin/sh` se tornar um processo filho):

```
$ echo $$
19877
$ ps auxf | grep 19877 | head -1
carol  19877  0.0  0.0   7448  3984 pts/25   Ss   21:17   0:00  \_ bash
$ exec /bin/sh
```

```
sh-5.0$ ps auxf | grep 19877 | head -1
carol   19877  0.0  0.0   7448  3896 pts/25   Ss   21:17   0:00  \_ /bin/sh
```

3. Siga estes passos para explorar os eventos do kernel e o gerenciamento dinâmico de dispositivos do udev:
   - Conecte um drive USB ao seu computador. Rode o `dmesg` e preste atenção às últimas linhas. Qual a linha mais recente?

     Deve aparecer algo semelhante a `[ 1967.700468] sd 6:0:0:0: [sdb] Attached SCSI removable disk`.

   - Tendo em mente a saída do comando anterior, execute `ls /dev/sd*` e verifique se seu drive USB aparece na lista. Qual a saída?

     Dependendo do número de dispositivos conectados ao sistema, deve aparecer algo semelhante a `/dev/sda /dev/sda1 /dev/sdb /dev/sdb1 /dev/sdb2`. Em nosso caso, encontramos nosso drive USB (`/dev/sdb`) e suas duas partições (`/dev/sdb1` e `/dev/sdb2`).

   - Agora, remova o drive USB e execute `dmesg` outra vez. O que diz a linha mais recente?

     Deve aparecer algo semelhante a `[ 2458.881695] usb 1-9: USB disconnect, device number 6`.

   - Execute `ls /dev/sd*` novamente e verifique se o dispositivo sumiu da lista. Qual a saída?

     Em nosso caso, `/dev/sda /dev/sda1`.

4.4 Seu Computador na Rede

Referência ao LPI objectivo

Linux Essentials version 1.6, Exam 010, Objective 4.4

Peso

2

Áreas chave de conhecimento

- Internet, rede, roteadores
- Consultando configuração do cliente DNS
- Consultando configuração da rede

Segue uma lista parcial de arquivos, termos e utilitários utilizados

- `route`, `ip route show`
- `ifconfig`, `ip addr show`
- `netstat`, `ss`
- `/etc/resolv.conf`, `/etc/hosts`
- IPv4, IPv6
- `ping`
- `host`

4.4 Lição 1

Certificação:	Linux Essentials
Versão:	1.6
Tópico:	4 O sistema operacional Linux
Objetivo:	4.4 Seu computador na rede
Lição:	1 de 1

Introdução

No mundo de hoje, qualquer tipo de dispositivo de computação troca informações por meio de redes. No cerne do conceito das redes de computadores estão as conexões físicas entre um dispositivo e seus pares. Essas conexões são chamadas *links* e constituem a conexão mais básica entre dois dispositivos diferentes. Os links podem ser estabelecidos através de vários meios, como cabos de cobre, fibras óticas, ondas de rádio ou lasers.

Cada link está conectado a uma interface de um dispositivo. Cada dispositivo pode ter múltiplas interfaces e, portanto, estar conectado a diversos links. Através desses links, os computadores podem formar uma rede: uma pequena comunidade de dispositivos capazes de se conectar diretamente. Há inúmeros exemplos dessas redes no mundo. Para poder se comunicar além do alcance de uma camada de link, os dispositivos usam roteadores. Pense nas redes da camada de link como ilhas conectadas entre si por roteadores — assim como pontes que as informações precisam atravessar para chegar a um dispositivo que faz parte de uma ilha remota.

Esse modelo leva a diversas camadas de rede diferentes:

Camada de link

Responsável pela comunicação entre dispositivos conectados diretamente.

Camada de rede

Gerencia o roteamento para fora das redes individuais e o endereçamento único de dispositivos além de uma única camada de link.

Camada de aplicativo

Permite que programas individuais se conectem uns aos outros.

Quando foram inventadas, as redes de computadores usavam os mesmos métodos de comunicação que os telefones, ou seja, eram comutadas por circuitos. Isso significa que um link dedicado e direto tinha de ser formado entre dois nós para que eles pudessem se comunicar. Esse método funcionava bem, mas exigia todo o espaço em um determinado link para que apenas dois hosts se comunicassem.

Depois de algum tempo, as redes de computadores passaram a usar a *comutação de pacotes*. Nesse método, os dados são agrupados com um cabeçalho contendo informações sobre a origem e o destino das informações. As informações do conteúdo estão contidas nessa estrutura e são enviadas pelo link para o destinatário indicado no cabeçalho. Isso permite que diversos dispositivos compartilhem um único link e se comuniquem quase simultaneamente.

Comunicação na camada de link

A função de qualquer pacote é transportar informações da origem até o destino através de um link que conecta os dois dispositivos. Esses dispositivos precisam de uma maneira de se identificar. Esse é o objetivo de um *endereço de camada de link*. Em uma rede Ethernet, os endereços de *Controle de Acesso de Mídia* (Media Access Control ou MAC) são usados para identificar dispositivos individuais. Um endereço MAC consiste em 48 bits. Eles não são necessariamente únicos globalmente e não podem ser usados para endereçar pontos externos ao link atual. Portanto, esses endereços não podem ser usados para rotear pacotes para outros links. O destinatário de um pacote verifica se o endereço de destino corresponde à sua própria camada de link e, se for o caso, processa o pacote. Caso contrário, o pacote será descartado. A exceção a essa regra são *pacotes de broadcast* (um pacote enviado a todos em uma determinada rede local), que sempre são aceitos.

O comando `ip link show` exibe uma lista de todas as interfaces de rede disponíveis e seus endereços da camada de link, além de outras informações sobre o tamanho máximo do pacote:

```
$ ip link show
```

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode DEFAULT
group default qlen 1000
    link/ether 00:0c:29:33:3b:25 brd ff:ff:ff:ff:ff:ff
```

A saída acima mostra que o dispositivo tem duas interfaces, `lo` e `ens33`. `lo` é o *dispositivo de loopback* e tem o endereço MAC `00:00:00:00:00:00`, ao passo que `ens33` é uma interface Ethernet com o endereço MAC `00:0c:29:33:3b:25`.

Rede IPv4

Para visitar sites como o Google ou o Twitter, verificar emails ou permitir que empresas se conectem entre si, os pacotes precisam ser capazes de passar de uma camada de link para outra. Freqüentemente, essas redes são conectadas indiretamente, com diversas camadas de link intermediárias que os pacotes devem atravessar para alcançar seu destino real.

Os endereços de camada de link de uma interface de rede não podem ser usados fora da rede específica em que está a camada de link. Como esse endereço não tem significado para os dispositivos de camada de link de outras redes, é necessária uma forma de endereços globalmente exclusivos para implementar o roteamento. Esse esquema de endereçamento, juntamente com o conceito geral de roteamento, é implementado pelo *Protocolo de Internet* (Internet Protocol ou IP).

NOTE

Um *protocolo* é um conjunto de procedimentos que permite que todas as partes que seguem o protocolo sejam compatíveis entre si. Um protocolo pode ser visto como a definição de um padrão. Em computação, o Internet Protocol é um padrão aceito por todos para que diferentes dispositivos produzidos por diferentes fabricantes possam se comunicar.

Endereços IPv4

Os endereços IP, como os endereços MAC, são uma maneira de indicar de onde vem um pacote de dados e para onde ele está indo. IPv4 era o protocolo original. Os endereços IPv4 têm 32 bits de largura, fornecendo um número máximo teórico de 4.294.967.296 endereços. No entanto, o número desses endereços que são utilizáveis pelos dispositivos é muito menor, pois alguns intervalos são reservados para casos de uso especiais, como endereços de broadcast (usados para alcançar todos os participantes de uma rede específica), endereços multicast (semelhantes aos endereços de broadcast, mas o dispositivo deve sintonizá-lo, como um rádio) ou endereços reservados para uso privado.

Em seu formato legível para humanos, os endereços IPv4 são representados como quatro grupos de dígitos separados por um ponto. Cada grupo pode variar de `0` a `255`. Por exemplo, veja o seguinte endereço IP:

```
192.168.0.1
```

Tecnicamente, cada um desses grupos de dígitos representa oito bits individuais. Assim, esse endereço também pode ser escrito desta maneira:

```
11000000.10101000.00000000.00000001
```

Na prática, usamos uma notação decimal como a mostrada acima. No entanto, a representação bit a bit ainda é importante para entender as sub-redes.

Sub-redes IPv4

Para oferecer suporte ao roteamento, os endereços IP podem ser divididos em duas partes: uma parte da rede e uma do host. A parte da rede identifica a rede na qual o dispositivo está e é usada para rotear pacotes para essa rede. A parte do host é usada para identificar especificamente um determinado dispositivo em uma rede e para entregar o pacote ao seu destinatário específico, uma vez atingida a sua camada de link.

Os endereços IP podem ser divididos em partes de sub-rede e host em qualquer ponto. A chamada *máscara de sub-rede* define onde essa divisão acontece. Vamos reconsiderar a representação binária do endereço IP do exemplo anterior:

```
11000000.10101000.00000000.00000001
```

Para esse endereço IP, a máscara de sub-rede define cada bit que pertence à parte da rede como `1` e cada bit que pertence à parte do host como `0`:

```
11111111.11111111.11111111.00000000
```

Na prática, a máscara de rede é escrita na notação decimal:

```
255.255.255.0
```

Isso significa que essa rede varia de `192.168.0.0` a `192.168.0.255`. Note que os três primeiros

números, que possuem todos os bits definidos na máscara de rede, permanecem inalterados nos endereços IP.

Por fim, existe uma notação alternativa para a máscara de sub-rede, chamada *Classless Inter-Domain Routing* (CIDR). Essa notação indica somente quantos bits estão definidos na máscara de sub-rede e adiciona esse número ao endereço IP. No exemplo, 24 dos 32 bits são definidos como `1` na máscara de sub-rede. Isso pode ser expresso na notação CIDR como `192.168.0.1/24`

Endereços IPv4 privados

Como mencionado anteriormente, determinadas seções do endereço IPv4 são reservadas para casos de uso especiais. Um desses casos de uso são atribuições de endereços privados. As seguintes sub-redes estão reservadas para endereçamento privado:

- `10.0.0.0/8`
- `172.16.0.0/12`
- `192.168.0.0/16`

Os endereços dessas sub-redes podem ser usados por qualquer pessoa. No entanto, essas sub-redes não podem ser roteadas na internet pública, pois são potencialmente usadas por várias redes ao mesmo tempo.

Hoje, a maioria das redes usa esses endereços internos. Eles permitem a comunicação interna sem a necessidade de qualquer atribuição de endereço externo. A maioria das conexões à Internet atualmente vem apenas com um único endereço IPv4 externo. Ao encaminhar pacotes para a Internet, os roteadores mapeiam todos os endereços internos para esse único endereço IP externo. Esse processo se chama *Network Address Translation* (NAT). O caso especial do NAT, em que um roteador mapeia endereços internos para um único endereço IP externo, às vezes é chamado de *masquerading*. Ele permite que qualquer dispositivo na rede interna estabeleça novas conexões com qualquer endereço IP global na internet.

NOTE Com o masquerading, os dispositivos internos não podem ser referenciados na Internet, pois não possuem um endereço válido globalmente. No entanto, esse não é um recurso de segurança. Mesmo ao usar masquerading, um firewall permanece necessário.

Configuração do endereço IPv4

Existem duas maneiras principais de configurar endereços IPv4 em um computador: atribuir os endereços manualmente ou usar o protocolo de configuração dinâmica de host (*Dynamic Host Configuration Protocol* ou DHCP) para configuração automática.

Ao usar o DHCP, um servidor central controla quais endereços são entregues a quais dispositivos. O servidor também pode fornecer aos dispositivos outras informações sobre a rede, como os endereços IP dos servidores DNS, o endereço IP do roteador padrão ou, no caso de configurações mais complicadas, iniciar um sistema operacional a partir da rede. O DHCP está ativado por padrão em muitos sistemas; portanto, você provavelmente já terá um endereço IP quando estiver conectado a uma rede.

Também é possível adicionar endereços IP manualmente a uma interface usando o comando `ip addr add`. Aqui, adicionamos o endereço `192.168.0.5` à interface `ens33`. A rede usa a máscara de rede `255.255.255.0`, que equivale a `/24` em notação CIDR:

```
$ sudo ip addr add 192.168.0.5/255.255.255.0 dev ens33
```

Agora, podemos verificar se o endereço foi adicionado, usando o comando `ip addr show`:

```
$ ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
25: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen
1000
    link/ether 00:0c:29:33:3b:25 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.5/24 192.168.0.255 scope global ens33
       valid_lft forever preferred_lft forever
    inet6 fe80::010c:29ff:fe33:3b25/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

A saída acima mostra a interface `lo` e a interface `ens33` com seu endereço atribuído com o comando acima.

Para verificar se um dispositivo está acessível, usamos o comando `ping`. Ele envia um tipo especial de mensagem chamado *echo request* (solicitação de eco), na qual o remetente solicita uma resposta ao destinatário. Se a conexão entre os dois dispositivos puder ser estabelecida com sucesso, o destinatário retornará uma resposta de eco, confirmando a conexão:

```
$ ping -c 3 192.168.0.1
PING 192.168.0.1 (192.168.0.1) 56(84) bytes of data.
64 bytes from 192.168.0.1: icmp_seq=1 ttl=64 time=2.16 ms
```

```
64 bytes from 192.168.0.1: icmp_seq=2 ttl=64 time=1.85 ms
64 bytes from 192.168.0.1: icmp_seq=3 ttl=64 time=3.41 ms

--- 192.168.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 5ms
rtt min/avg/max/mdev = 1.849/2.473/3.410/0.674 ms
```

O parâmetro `-c 3` faz com que o `ping` pare de ser executado após o envio de três solicitações de eco. Caso contrário, o `ping` continua a rodar para sempre e precisa ser interrompido pressionando `Ctrl` + `C`.

Roteamento IPv4

O roteamento é o processo no qual um pacote passa da rede de origem para a rede de destino. Cada dispositivo mantém uma tabela de roteamento contendo informações sobre qual rede IP pode ser acessada diretamente através do vínculo do dispositivo com a camada de link e qual rede IP pode ser alcançada com a transmissão de pacotes para um roteador. Por fim, uma *rota padrão* define um roteador que recebe todos os pacotes que não correspondem a nenhuma outra rota.

Ao estabelecer uma conexão, o dispositivo procura o endereço IP do destino em sua tabela de roteamento. Se uma entrada corresponder ao endereço, o pacote será enviado à camada de link respectiva ou repassado ao roteador indicado na tabela de roteamento.

Os próprios roteadores também mantêm tabelas de roteamento. Ao receber um pacote, um roteador procura o endereço de destino em sua própria tabela e envia o pacote para o roteador seguinte. Isso é repetido até que o pacote chegue ao roteador na rede de destino. Cada roteador envolvido nessa jornada é chamado de *hop*. O último roteador encontra um link conectado direto para o endereço de destino em sua tabela de roteamento e envia os pacotes para a interface do destino.

A maioria das redes domésticas conta com apenas uma saída, o roteador enviado pelo *provedor de internet* (ISP). Nesse caso, o dispositivo apenas encaminha todos os pacotes que não são para a rede interna diretamente para o roteador doméstico, que o envia então ao roteador do provedor para encaminhamento adicional. Esse é um exemplo de rota padrão.

O comando `ip route show` lista a tabela atual de roteamento IPv4:

```
$ ip route show
127.0.0.0/8 via 127.0.0.1 dev lo0
192.168.0.0/24 dev ens33 scope link
```

Para adicionar uma rota padrão, basta ter o endereço interno do roteador que será o gateway padrão. Se, por exemplo, o roteador tiver o endereço `192.168.0.1`, o seguinte comando o configurará como uma rota padrão:

```
$ sudo ip route add default via 192.168.0.1
```

Para confirmar, execute `ip route show` novamente:

```
$ ip route show
default via 192.168.0.1 dev ens33
127.0.0.0/8 via 127.0.0.1 dev lo0
192.168.0.0/24 dev ens33 scope link
```

Rede IPv6

O IPv6 foi projetado para colmatar as deficiências do IPv4, principalmente a falta de endereços conforme mais e mais dispositivos iam sendo postos online. No entanto, o IPv6 também inclui outros recursos, como novos protocolos para a configuração automática de redes. Em vez de 32 bits por endereço, o IPv6 usa 128. Isso permite aproximadamente 2^{128} endereços. No entanto, como o IPv4, o número de endereços utilizáveis globalmente exclusivos é muito menor porque algumas seções da alocação são reservadas para outros usos. Esse grande número de endereços significa que existem endereços públicos mais do que suficientes para todos os dispositivos atualmente conectados à Internet, mais muitos outros, reduzindo assim a necessidade de masquerading e os problemas gerados por ele, como o atraso na tradução e a impossibilidade de conectar-se diretamente a dispositivos mascarados.

Endereços IPv6

Quando escritos, os endereços usam 8 grupos de 4 dígitos hexadecimais, separados por dois pontos:

```
2001:0db8:0000:abcd:0000:0000:0000:7334
```

NOTE Os dígitos hexadecimais vão de `0` a `f`, de forma que cada dígito pode conter um dentre 16 valores diferentes.

Para facilitar, os zeros à esquerda de cada grupo podem ser removidos quando escritos, mas cada grupo deve conter pelo menos um dígito:

```
2001:db8:0:abcd:0:0:0:7334
```

Quando diversos grupos contendo apenas zeros aparecem diretamente um após o outro, podem ser substituídos por '::':

```
2001:db8:0:abcd::7334
```

Todavia, isso só pode ser feito uma vez em cada endereço.

Prefixo IPv6

Os primeiros 64 bits de um endereço IPv6 são conhecidos como o *prefixo de roteamento*. O prefixo é usado pelos roteadores para determinar a qual rede um dispositivo pertence e, portanto, em qual caminho os dados precisam ser enviados. A sub-rede sempre acontece dentro do prefixo. Os provedores de internet geralmente distribuem prefixos /48 ou /58 aos seus clientes, deixando 16 ou 8 bits para sua sub-rede interna.

Existem três tipos principais de prefixos no IPv6:

Endereço Global Unicast

Nele, o prefixo é atribuído a partir dos blocos reservados aos endereços globais. Esses endereços são válidos em toda a internet.

Endereço Local Unicast

Pode não ser roteado na internet. No entanto, pode ser roteado internamente dentro de uma organização. Esses endereços são usados dentro de uma rede para garantir que os dispositivos tenham um endereço mesmo quando não houver conexão com a Internet. Equivalem aos intervalos de endereços privados do IPv4. Os primeiros 8 bits são sempre `fc` ou `fd`, seguidos por 40 bits gerados aleatoriamente.

Endereço Link-Local

São válidos somente em um link específico. Toda interface de rede compatível com IPv6 possui um desses endereços, começando com `fe80`. Esses endereços são usados internamente pelo IPv6 para solicitar endereços adicionais usando a configuração automática e para encontrar outros computadores na rede usando o protocolo Neighbor Discovery.

Identificador da interface IPv6

Enquanto o prefixo determina em qual rede um dispositivo reside, o identificador de interface é

usado para enumerar os dispositivos em uma rede. Os últimos 64 bits de um endereço IPv6 formam o identificador da interface, assim como os últimos bits de um endereço IPv4.

Quando um endereço IPv6 é atribuído manualmente, o identificador da interface é definido como parte do endereço. Ao usar a configuração automática de endereço, o identificador de interface é escolhido aleatoriamente ou derivado do endereço da camada de link do dispositivo. Isso faz com que uma variação do endereço da camada de link apareça no endereço IPv6.

Configuração do endereço IPv6

Como o IPv4, o endereço IPv6 pode ser atribuído manualmente ou automaticamente. No entanto, o IPv6 possui dois tipos diferentes de configuração automatizada: DHCPv6 e *Stateless Address Autoconfiguration* (SLAAC).

O SLAAC é o mais fácil dos dois métodos automatizados, sendo incorporado no padrão IPv6. As mensagens usam o novo *Neighbor Discovery Protocol*, que permite que os dispositivos se encontrem e solicitem informações a respeito de uma rede. Essas informações são enviadas pelos roteadores e podem conter prefixos IPv6 que os dispositivos podem usar combinando-os com um identificador de interface de sua escolha, desde que o endereço resultante ainda não esteja em uso. Os dispositivos não fornecem feedback ao roteador sobre os endereços criados.

O DHCPv6, por outro lado, é o DHCP atualizado para funcionar com as alterações do IPv6. Ele permite um controle mais preciso das informações entregues aos clientes, como permitir que o mesmo endereço seja sempre entregue ao mesmo cliente e enviar mais opções para o cliente do que o SLAAC. Com o DHCPv6, os clientes precisam obter o consentimento explícito de um servidor DHCP para usar um endereço.

O método para atribuir manualmente um endereço IPv6 a uma interface é o mesmo do IPv4:

```
$ sudo ip addr add 2001:db8:0:abcd:0:0:0:7334/64 dev ens33
```

Para confirmar se a atribuição funcionou, usamos o mesmo comando, `ip addr show`:

```
$ ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
25: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen
```

```
1000
    link/ether 00:0c:29:33:3b:25 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.5/24 192.168.0.255 scope global ens33
       valid_lft forever preferred_lft forever
    inet6 fe80::010c:29ff:fe33:3b25/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
    inet6 2001:db8:0:abcd::7334/64 scope global
       valid_lft forever preferred_lft forever
```

Aqui também vemos o endereço link-local `fe80::010c:29ff:fe33:3b25/64`.

Como no caso do IPv4, o comando `ping` também pode ser usado para confirmar a acessibilidade dos dispositivos através do IPv6:

```
$ ping 2001:db8:0:abcd::1
PING 2001:db8:0:abcd::1(2001:db8:0:abcd::1) 56 data bytes
64 bytes from 2001:db8:0:abcd::1: icmp_seq=1 ttl=64 time=0.030 ms
64 bytes from 2001:db8:0:abcd::1: icmp_seq=2 ttl=64 time=0.040 ms
64 bytes from 2001:db8:0:abcd::1: icmp_seq=3 ttl=64 time=0.072 ms

--- 2001:db8:0:abcd::1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 43ms
rtt min/avg/max/mdev = 0.030/0.047/0.072/0.018 ms
```

NOTE Em certos sistemas Linux, o `ping` não suporta IPv6. Esses sistemas incluem um comando dedicado `ping6` para isso.

Para confirmar o endereço link-local, use `ping` novamente. Mas como todas as interfaces usam o prefixo `fe80::/64`, a interface correta deve ser especificada junto com o endereço:

```
$ ping6 -c 1 fe80::010c:29ff:fe33:3b25%ens33
PING fe80::010c:29ff:fe33:3b25(fe80::010c:29ff:fe33:3b25) 56 data bytes
64 bytes from fe80::010c:29ff:fe33:3b25%ens33: icmp_seq=1 ttl=64 time=0.049 ms

--- fe80::010c:29ff:fe33:3b25 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.049/0.049/0.049/0.000 ms
```

DNS

Os endereços IP são difíceis de memorizar e não são lá muito atraentes quando estamos tentando

comercializar um serviço ou produto. É aqui que o *Domain Name System* (sistema de nomes de domínio) entra em ação. Em sua forma mais simples, o DNS é uma lista telefônica que mapeia nomes de domínio memoráveis e amigáveis, como `exemplo.com`, para endereços IP. Quando, por exemplo, um usuário navega até um site, ele insere o nome do host DNS como parte da URL. O navegador web envia o nome de DNS para o resolvedor DNS configurado. Esse resolvedor de DNS, por sua vez, descobrirá o endereço relacionado àquele domínio. O resolvedor responde com essa informação e o navegador tenta acessar o servidor web nesse endereço IP.

Os resolvedores usados pelo Linux para procurar dados DNS estão no arquivo de configuração `/etc/resolv.conf`:

```
$ cat /etc/resolv.conf
search lpi
nameserver 192.168.0.1
```

Quando o resolvedor realiza uma pesquisa de nome, ele primeiro verifica o arquivo `/etc/hosts` para ver se encontra um endereço para o nome solicitado. Se for o caso, ele retorna esse endereço e não entra em contato com o DNS. Isso permite que os administradores de rede forneçam a resolução de nomes sem a necessidade de configurar um servidor DNS completo. Cada linha desse arquivo contém um endereço IP seguido por um ou mais nomes:

```
127.0.0.1          localhost.localdomain   localhost
::1                localhost.localdomain   localhost
192.168.0.10       server
2001:db8:0:abcd::f server
```

Para realizar uma pesquisa no DNS, use o comando `host`:

```
$ host learning.lpi.org
learning.lpi.org has address 208.94.166.198
```

O comando `dig` oferece informações mais detalhadas:

```
$ dig learning.lpi.org
; <<>> DiG 9.14.3 <<>> learning.lpi.org
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 21525
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
```

```
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: 2ac55879b1adef30a93013705d3306d2128571347df8eadf (bad)
;; QUESTION SECTION:
;learning.lpi.org.      IN  A

;; ANSWER SECTION:
learning.lpi.org.   550 IN  A   208.94.166.198

;; Query time: 3 msec
;; SERVER: 192.168.0.1#53(192.168.0.1)
;; WHEN: Sat Jul 20 14:20:21 EST 2019
;; MSG SIZE  rcvd: 89
```

Aqui também vemos o nome dos tipos de registro DNS, neste caso `A` para IPv4.

Sockets

Um *socket* é um terminal de comunicação para dois programas conversando entre si. Se o socket estiver conectado por uma rede, os programas poderão ser executados em dispositivos diferentes, como um navegador web no laptop de um usuário e um servidor web no data center de uma empresa.

Existem três tipos principais de sockets:

Sockets Unix

Conecta os processos em execução no mesmo dispositivo.

Sockets UDP (User Datagram Protocol)

Conectam aplicativos usando um protocolo rápido, mas pouco resistente.

Sockets TCP (Transmission Control Protocol)

Mais confiáveis que os sockets UDP; por exemplo, eles confirmam a recepção de dados.

Os sockets Unix só podem conectar aplicativos em execução no mesmo dispositivo. Já os sockets TCP e UDP podem se conectar através de uma rede. O TCP permite um fluxo de dados que sempre chegam na ordem exata em que foram enviados. O UDP meio que se guia sozinho; o pacote é enviado, mas a forma de entrega na outra extremidade não é garantida. No entanto, o UDP não tem a mesma sobrecarga do TCP, sendo assim perfeito para aplicações de baixa latência, como videogames online.

O TCP e o UDP usam portas para endereçar vários sockets no mesmo endereço IP. Embora a porta

de origem de uma conexão geralmente seja aleatória, as portas de destino são padronizadas para um serviço específico. O HTTP, por exemplo, geralmente é hospedado na porta 80; o HTTPS é executado na porta 443. O SSH, um protocolo para fazer login com segurança em um sistema Linux remoto, escuta na porta 22.

O comando `ss` permite que um administrador investigue todos os sockets em um computador Linux. Ele mostra tudo, desde o endereço de origem até o endereço de destino e o tipo. Um de seus melhores recursos é o uso de filtros que permitem monitorar os sockets em qualquer estado de conexão que se queira. O `ss` pode ser executado com um conjunto de opções, além de uma expressão de filtro para limitar as informações mostradas.

Quando executado sem nenhuma opção, o comando exibe uma lista de todos os sockets estabelecidos. A opção `-p` inclui informações sobre o processo que está usando cada socket. A opção `-s` mostra um resumo dos sockets. Há muitas outras opções disponíveis para esta ferramenta, mas algumas das principais são `-4` e `-6` para restringir o protocolo IP a IPv4 ou IPv6 respectivamente; `-t` e `-u` permitem que o administrador selecione sockets TCP ou UDP e `-l` exibe apenas o socket que está atento às novas conexões.

O comando a seguir, por exemplo, lista todos os sockets TCP atualmente em uso:

```
$ ss -t
State       Recv-Q  Send-Q    Local Address:Port      Peer Address:Port
ESTAB       0       0           192.168.0.5:49412      192.168.0.1:https
ESTAB       0       0           192.168.0.5:37616      192.168.0.1:https
ESTAB       0       0           192.168.0.5:40114      192.168.0.1:https
ESTAB       0       0           192.168.0.5:54948      192.168.0.1:imap
...
```

Exercícios Guiados

1. Um engenheiro de rede precisa atribuir dois endereços IP à interface `ens33` de um host, um endereço IPv4 (`192.168.10.10/24`) e um endereço IPv6 (`2001:0:0:abcd:0:8a2e:0370:7334/64`). Quais comandos ele deve usar para fazer isso?

2. Quais endereços da lista abaixo são privados?

`192.168.10.1`	
`120.56.78.35`	
`172.16.57.47`	
`10.100.49.162`	
`200.120.42.6`	

3. Qual item você adicionaria ao arquivo de hosts para atribuir `192.168.0.15` a `example.com`?

4. Que efeito teria o comando a seguir?

```
sudo ip -6 route add default via 2001:db8:0:abcd::1
```

Exercícios Exploratórios

1. Cite o tipo de registro de DNS usado para atender às seguintes solicitações:
   - Dados textuais
   - Pesquisa reversa de endereço IP
   - Um domínio que não tem endereço próprio e precisa de outro domínio para obter essas informações
   - Servidor de email
2. O Linux tem um recurso chamado bridging, o que ele faz e qual sua utilidade?
3. Qual opção precisa ser fornecida ao comando `ss` para visualizar todos os sockets UDP estabelecidos?
4. Qual comando exibe um resumo de todos os sockets em execução em um dispositivo Linux?
5. A saída a seguir foi gerada pelo comando do exercício anterior. Quantos sockets TCP e UDP estão ativos?

```
Total: 978 (kernel 0)
TCP:   4 (estab 0, closed 0, orphaned 0, synrecv 0, timewait 0/0), ports 0

Transport Total     IP        IPv6
*     0         -         -
RAW   1         0         1
UDP   7         5         2
TCP   4         3         1
INET      12        8         4
FRAG      0         0         0
```

Resumo

Este tópico trata do uso de redes em seu computador Linux. Primeiro, aprendemos sobre os vários níveis de rede:

- A camada de link, que conecta os dispositivos diretamente.
- A camada de rede, que faz o roteamento entre as redes e um espaço de endereço global.
- A camada de aplicativo, na qual os aplicativos se conectam.

Vimos como o IPv4 e o IPv6 são usados para endereçar computadores individuais e como o TCP e o UDP enumeram os sockets usados pelos aplicativos para se conectar. Também aprendemos como o DNS é usado para resolver nomes para endereços IP.

Comandos usados nos exercícios:

`dig`
: Consulta informações de DNS e fornece dados detalhados sobre as consultas e respostas de DNS.

`host`
: Consulta informações de DNS e fornece uma saída condensada.

`ip`
: Configura o uso da rede no Linux, incluindo interfaces de rede, endereços e roteamento.

`ping`
: Testa a conectividade com um dispositivo remoto.

`ss`
: Mostra informações sobre sockets.

Respostas aos Exercícios Guiados

1. Um engenheiro de rede precisa atribuir dois endereços IP à interface `ens33` de um host, um endereço IPv4 (`192.168.10.10/24`) e um endereço IPv6 (`2001:0:0:abcd:0:8a2e:0370:7334/64`). Quais comandos ele deve usar para fazer isso?

```
sudo ip addr add 192.168.10.10/24 dev ens33
sudo ip addr add 2001:0:0:abcd:0:8a2e:0370:7334/64 dev ens33
```

2. Quais endereços da lista abaixo são privados?

`192.168.10.1`	X
`120.56.78.35`	
`172.16.57.47`	X
`10.100.49.162`	X
`200.120.42.6`	

3. Qual item você adicionaria ao arquivo de hosts para atribuir `192.168.0.15` a `example.com`?

```
192.168.0.15  example.com
```

4. Que efeito teria o comando a seguir?

```
sudo ip -6 route add default via 2001:db8:0:abcd::1
```

   Ele adicionaria à tabela uma rota padrão que envia todo o tráfego IPv6 ao roteador com o endereço interno `2001:db8:0:abcd::1`.

Respostas aos Exercícios Exploratórios

1. Cite o tipo de registro de DNS usado para atender às seguintes solicitações:
   - Dados textuais

     TXT

   - Pesquisa reversa de endereço IP

     PTR

   - Um domínio que não tem endereço próprio e precisa de outro domínio para obter essas informações

     CNAME

   - Servidor de email

     MX

2. O Linux tem um recurso chamado bridging, o que ele faz e qual sua utilidade?

   Um bridge conecta múltiplas interfaces de rede. Todas as interfaces conectadas a um bridge podem se comunicar como se estivessem conectadas à mesma camada de link: Todos os dispositivos usam endereços IP da mesma sub-rede e não requerem um roteador para se conectar entre si.

3. Qual opção precisa ser fornecida ao comando `ss` para visualizar todos os sockets UDP estabelecidos?

   A opção `-u` mostra todos os sockets UDP estabelecidos.

4. Qual comando exibe um resumo de todos os sockets em execução em um dispositivo Linux?

   O comando `ss -s` mostra um resumo de todos os sockets

5. A saída a seguir foi gerada pelo comando do exercício anterior. Quantos sockets TCP e UDP estão ativos?

```
Total: 978 (kernel 0)
TCP:   4 (estab 0, closed 0, orphaned 0, synrecv 0, timewait 0/0), ports 0

Transport Total     IP        IPv6
```

```
*     0         -         -
RAW   1         0         1
UDP   7         5         2
TCP   4         3         1
INET      12        8         4
FRAG      0         0         0
```

11 sockets TCP e UDP estão ativos.

Linux Professional Institute

Tópico 5: Segurança e Permissões de Arquivos

5.1 Segurança Básica e Identificação de Tipos de Usuários

Referência ao LPI objectivo

Linux Essentials version 1.6, Exam 010, Objective 5.1

Peso

2

Áreas chave de conhecimento

- Root e Usuários padrão
- Usuários do sistema

Segue uma lista parcial de arquivos, termos e utilitários utilizados

- `/etc/passwd`, `/etc/shadow`, `/etc/group`
- `id`, `last`, `who`, `w`
- `sudo`, `su`

5.1 Lição 1

Certificação:	Linux Essentials
Versão:	1.6
Tópico:	5 Segurança e permissões de arquivos
Objetivo:	5.1 Segurança básica e identificação de tipos de usuários
Lição:	1 de 1

Introdução

Esta lição abordará a terminologia básica relacionada a contas, controles de acesso e segurança dos sistemas Linux locais, as ferramentas da interface da linha de comando (CLI) em um sistema Linux para controles de acesso básicos e os arquivos essenciais para oferecer suporte a contas de usuários e grupos, incluindo os usados para aumentar os privilégios elementares.

A segurança básica nos sistemas Linux deriva-se dos controles de acesso do Unix que, apesar de terem quase cinquenta anos, são bastante eficazes em comparação com outros sistemas operacionais populares de linhagem muito mais recente. Mesmo certos sistemas operacionais populares baseados em Unix tendem a “tomar liberdades” com o argumento da “facilidade de acesso”, coisa que o Linux não faz.

Os ambientes e interfaces modernos do Linux simplificam a criação e o gerenciamento de usuários e frequentemente automatizam a atribuição de controles de acesso quando um usuário faz login — por exemplo, permitindo acesso ao monitor, áudio e outros serviços — sem a necessidade de praticamente nenhuma intervenção manual do administrador do sistema. No entanto, é importante entender os conceitos básicos por trás de um sistema operacional Linux.

Contas

A segurança engloba muitos conceitos, sendo um dos mais comuns o conceito geral de controles de acesso. Antes de tratar dos controles de acesso a arquivos, como propriedade e permissões, é necessário compreender os conceitos básicos relacionados às *contas* de usuário do Linux, que são divididas em vários tipos.

Todo usuário de um sistema Linux possui uma conta associada que, além das informações de login (como nome de usuário e senha), também define como e onde o usuário pode interagir com o sistema. Os privilégios e controles de acesso definem os "limites" dentro dos quais cada usuário pode operar.

Identificadores (UIDs/GIDs)

Os *Identificadores de Usuário* e *Grupos* (UIDs/GIDs) são as referências básicas, numeradas, às contas. As implementações iniciais eram números inteiros limitados de 16 bits (valores de 0 a 65535), mas os sistemas do século XXI suportam UIDs e GIDs de 64 bits. Os usuários e grupos são enumerados de forma independente, de modo que o mesmo ID pode representar tanto um usuário quanto um grupo.

Todo usuário possui não apenas um UID, mas também um *GID primário*. O GID primário de um usuário pode ser exclusivo apenas para esse usuário, não sendo usado por outros usuários. No entanto, o grupo pode ser compartilhado por diversos usuários. Além desses grupos primários, cada usuário também pode ser membro de outros grupos.

Por padrão, nos sistemas Linux, todo usuário é incluído em um grupo com o mesmo nome que seu nome de usuário e um GID idêntico a seu UID. Por exemplo, se criarmos um novo usuário chamado `newuser`, seu grupo por padrão também será `newuser`.

A conta de superusuário

No Linux, a conta do superusuário é `root`, que sempre terá o UID 0. O superusuário também pode ser chamado de *administrador do sistema* e dispõe de acesso e controle ilimitados sobre o sistema, incluindo outros usuários.

O grupo padrão do superusuário tem o GID `0` e também é chamado de `root`. O diretório inicial do superusuário é um diretório dedicado, de nível superior, `/root`, acessível apenas pelo próprio usuário `root`.

Contas de usuário padrão

Todas as contas que não sejam `root` são, tecnicamente, contas de usuário regulares, mas em um sistema Linux o termo coloquial *conta de usuário* geralmente indica uma conta de usuário "regular" (sem privilégios). Elas têm geralmente as seguintes propriedades, com algumas exceções:

- UIDs iniciando em 1000 (4 dígitos), embora alguns sistemas legados possam iniciar em 500.
- Um diretório inicial definido, geralmente um subdiretório de `/home`, dependendo da configuração local.
- Um shell de login definido. No Linux, o shell padrão é geralmente o *Bourne Again Shell* (`/bin/bash`), embora outros possam estar disponíveis.

Se uma conta de usuário não tiver um shell válido em seus atributos, o usuário não poderá abrir um shell interativo. Normalmente, `/sbin/nologin` é usado como um shell inválido. Isso pode ser proposital, no caso de o usuário ser autenticado apenas para serviços que excluam o console ou o acesso SSH; por exemplo, somente acesso FTP seguro (`sftp`).

NOTE Para evitar confusões, o termo *conta de usuário* será aplicado apenas a contas de usuário padrão ou regulares daqui para a frente. Usaremos *conta de sistema* para nos referir uma conta de usuário Linux pertencente ao tipo de conta de usuário do sistema.

Contas do sistema

As *contas de sistema* são tipicamente pré-criadas no momento da instalação do sistema. São destinadas a recursos, programas e serviços que não serão executados como superusuário. Em um mundo ideal, todos eles seriam funcionalidades do sistema operacional.

As contas de sistema variam, mas dentre seus atributos temos:

- Os UIDs normalmente são menores de 100 (2 dígitos) ou entre 500-1000 (3 dígitos).
- Não têm *nenhum* diretório inicial dedicado ou têm um diretório que normalmente não está em `/home`.
- Nenhum shell de login válido (tipicamente `/sbin/nologin`), com raras exceções.

A maioria das contas de sistema no Linux nunca faz login e não precisa de um shell definido em seus atributos. Muitos processos que pertencem e são executados por contas de sistema são transferidos para o seu próprio ambiente pelo gerenciamento do sistema, sendo executados com a conta de sistema especificada. Essas contas geralmente têm privilégios limitados ou, na maioria

das vezes, *nenhum* privilégio.

NOTE Do ponto de vista do LPI Linux Essentials, as contas de sistema têm UIDs <1000, com UIDs (e GIDs) de 2 ou 3 dígitos.

Em geral, as contas de sistema *não* devem ter um shell de login válido. O contrário representaria um problema de segurança na maioria dos casos.

Contas de serviço

As *contas de serviço* são criadas tipicamente quando instalamos e configuramos serviços. Como no caso das contas de sistema, elas são destinadas aos recursos, programas e serviços que não serão executados como superusuário.

Em muitas documentações, as contas de sistema e de serviço são semelhantes e intercambiáveis. Isso inclui a localização dos diretórios pessoais, que normalmente ficam fora de `/home`, quando definidos (as contas de serviço têm mais probabilidade de ter um diretório pessoal do que as contas de sistema) e a ausência de um shell de login válido. Embora não exista uma definição estrita, a principal diferença entre contas de sistema e de serviço está no UID/GID.

Conta de sistema

UID/GID <100 (2 dígitos) ou <500-1000 (3 digitos)

Conta de serviço

UID/GID >1000 (4+ dígitos), mas não uma conta de usuário “padrão” ou “regular” , uma conta para os serviços, com um UID/GID >1000 (4+ dígitos)

Algumas distribuições Linux ainda têm contas de serviço pré-reservadas com UID <100, e estas também podem ser consideradas contas de sistema, mesmo que não sejam criadas na instalação. Por exemplo, nas distribuições Linux baseadas no Fedora (incluindo o Red Hat), o usuário do servidor web Apache tem um UID (e GID) 48, claramente uma conta de sistema, apesar de contar com um diretório inicial (geralmente em `/usr/share/httpd` ou `/var/www/html/`).

NOTE Do ponto de vista do LPI Linux Essentials, as contas de sistema têm UIDs <1000 e as contas de usuário regulares têm UIDs >1000. Como as contas de usuário regulares são >1000, esses UIDs também podem incluir contas de serviço.

Shells de login e diretórios iniciais

Algumas contas têm um shell de login, enquanto outras não os têm por motivos de segurança, já que não requerem acesso interativo. O shell de login padrão na maioria das distribuições Linux é o *Bourne Again Shell*, ou `bash`, mas outros shells podem estar disponíveis, como o C Shell (`csh`),

Korn shell (`ksh`) ou Z shell (`zsh`), dentre outros.

O usuário pode alterar seu shell de login usando o comando `chsh`. Por padrão, o comando é executado no modo interativo e exibe um prompt perguntando qual shell deve ser usado. A resposta deve ser o caminho completo para o binário do shell, como mostrado abaixo:

```
$ chsh

Changing the login shell for emma
Enter the new value, or press ENTER for the default
    Login Shell [/bin/bash]: /usr/bin/zsh
```

Também é possível executar o comando no modo não interativo, com o parâmetro `-s` seguido pelo caminho para o binário, da seguinte forma:

```
$ chsh -s /usr/bin/zsh
```

A maioria das contas tem um diretório inicial definido. No Linux, esse é geralmente o único local em que a conta do usuário tem acesso garantido de gravação, com algumas exceções (por exemplo, áreas temporárias do sistema de arquivos). No entanto, algumas contas são configuradas propositalmente para não ter acesso de gravação, mesmo no próprio diretório pessoal, por motivos de segurança.

Obtendo informações sobre os usuários

Listar informações básicas de usuário é uma prática cotidiana comum em um sistema Linux. Em alguns casos, os usuários precisarão trocar de usuário e ganhar privilégios para concluir tarefas privilegiadas.

Os usuários comuns podem listar atributos e acessos na linha de comando usando os comandos abaixo. As informações básicas em contexto limitado não são uma operação privilegiada.

Para listar as informações atuais de um usuário na linha de comando, basta um comando de duas letras: `id`. A saída vai depender de seu ISD de login:

```
$ id
uid=1024(emma) gid=1024(emma) 1024(emma),20(games),groups=10240(netusers),20480(netadmin)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

Na listagem anterior, o usuário (`emma`) tem identificadores detalhados da seguinte maneira:

- `1024` = ISD de usuário (UID), seguido pelo nome de usuário (nome comum ou nome de login) entre parênteses.
- `1024` = o ID *primário* de grupo (GID), seguido pelo nome do grupo (nome comum) entre parênteses.
- Uma lista de GIDs adicionais (nomes de grupo) a que o usuário também pertence.

Para listar a última vez em que os usuários se logaram no sistema, usamos o comando `last`:

```
$ last
emma     pts/3        ::1              Fri Jun 14 04:28   still logged in
reboot   system boot  5.0.17-300.fc30. Fri Jun 14 04:03   still running
reboot   system boot  5.0.17-300.fc30. Wed Jun  5 14:32 - 15:19  (00:46)
reboot   system boot  5.0.17-300.fc30. Sat May 25 18:27 - 19:11  (00:43)
reboot   system boot  5.0.16-100.fc28. Sat May 25 16:44 - 17:06  (00:21)
reboot   system boot  5.0.9-100.fc28.x Sun May 12 14:32 - 14:46  (00:14)
root     tty2                         Fri May 10 21:55 - 21:55  (00:00)
    ...
```

As informações listadas nas colunas podem variar, mas dentre os itens relevantes na listagem anterior podemos destacar:

- Um usuário (`emma`) fez login através da rede (pseudo TTY `pts/3`) e ainda está logado.
- A hora da inicialização atual está listada, junto com o kernel. No exemplo acima, cerca de 25 minutos antes do usuário fazer login.
- O superusuário (`root`) se logou através de um console virtual (TTY `tty2`), brevemente, em meados de maio.

Uma variante do comando `last` é o comando `lastb`, que lista as últimas tentativas malsucedidas de login.

Os comandos `who` e `w` listam somente os logins ativos no sistema:

```
$ who
emma  pts/3        2019-06-14 04:28 (::1)

$ w
 05:43:41 up  1:40,  1 user,  load average: 0.25, 0.53, 0.51
USER     TTY        LOGIN@   IDLE   JCPU   PCPU WHAT
emma  pts/3     04:28    1:14m  0.04s  0.04s -bash
```

Algumas informações são comuns aos dois comandos. Por exemplo, um usuário (`emma`) está logado com um pseudo-dispositivo TTY (`pts/3`) e a hora de login é 04:28.

O comando `w` lista mais informações, incluindo o seguinte:

- A hora atual e há quanto tempo o sistema está ativo
- Quantos usuários estão conectados
- As *cargas médias* dos últimos 1, 5 e 15 minutos

E as informações adicionais para cada sessão de usuário ativo.

- Tempos selecionados e totais de uso da CPU (`IDLE`, `JCPU` e `PCPU`)
- O processo atual (`-bash`). O tempo total de uso da CPU desse processo é o último item (`PCPU`).

Ambos os comandos têm mais opções para listar várias informações adicionais.

Troca de usuário e aumento de privilégios

Em um mundo ideal, os usuários nunca precisariam de privilégios adicionais para concluir suas tarefas. O sistema sempre "funcionaria" e tudo estaria configurado para diversos tipos de acesso.

Felizmente para nós, o Linux — desde a instalação — já funciona dessa maneira para a maioria dos usuários que não são administradores de sistema, apesar de sempre seguir o modelo de segurança do *menor privilégio*.

No entanto, existem comandos que permitem aumentar os privilégios quando necessário. Dois dos mais importantes são `su` e `sudo`.

Atualmente, na maioria dos sistemas Linux, o comando `su` é usado apenas para conceder privilégios de root ao usuário padrão (se um nome de usuário não for especificado após o nome do comando). Embora possa ser usado para alternar para outro usuário, não é uma boa prática: os usuários devem fazer login a partir de outro sistema, pela rede, console físico ou terminal no sistema.

```
emma ~$ su -
Password:
root ~#
```

Após digitar a senha do superusuário (`root`), o usuário adquire um shell de superusuário (observe o `#` no final do prompt de comando) e se torna, para todos os efeitos, o superusuário (`root`).

Compartilhar senhas é uma péssima ideia para a segurança, e por isso deve haver pouquíssima necessidade do comando `su` em um sistema Linux moderno.

O cifrão (`$`) deve encerrar o prompt da linha de comando no shell de usuário não privilegiado, enquanto o símbolo da cerquilha (`#`) deve encerrar o prompt da linha de comando no shell do superusuário (`root`). É altamente recomendável que o caractere final de qualquer prompt *nunca* se afaste desse padrão "universalmente aceito", uma vez que essa nomenclatura é empregada em materiais de aprendizado, incluindo estes.

WARNING

Nunca mude para o superusuário (`root`) sem passar o parâmetro do shell de login (`-`). A menos que haja instruções explícitas do fornecedor do SO ou do software, quando o `su` for necessário sempre se deve executar `su -`, com pouquíssimas exceções. Os ambientes de usuário podem causar problemas e alterações indesejáveis de configuração quando usados no modo de privilégio total como superusuário.

Qual o grande problema de se usar `su` para passar ao modo superusuário (`root`)? Se a sessão de um usuário regular tiver sido comprometida, a senha do superusuário (`root`) poderia ser capturada. É aí que entra o "Switch User Do" (ou "Superuser Do"):

```
$ cat /sys/devices/virtual/dmi/id/board_serial
cat: /sys/devices/virtual/dmi/id/board_serial: Permission denied

$ sudo cat /sys/devices/virtual/dmi/id/board_serial
[sudo] password for emma:
/6789ABC/
```

Na listagem anterior, o usuário está tentando procurar o número de série da placa do sistema. No entanto, a permissão é negada, pois essas informações estão marcadas como privilegiadas.

Porém, usando o `sudo`, o usuário digita sua própria senha para se autenticar. Se ele tiver sido autorizado na configuração `sudoers` para executar esse comando com privilégio, com as opções permitidas, ele funcionará.

TIP

Por padrão, o primeiro comando autorizado `sudo` autenticará os comandos `sudo` subsequentes por um período (muito curto) de tempo. Isso pode ser configurado pelo administrador do sistema.

Arquivos de controle de acesso

Quase todos os sistemas operacionais têm um conjunto de locais usados para armazenar os

controles de acesso. No Linux, esses são tipicamente arquivos de texto localizados no diretório `/etc`, que é onde costumam ser armazenados os arquivos de configuração do sistema. Por padrão, esse diretório é legível por todos os usuários no sistema, mas modificável apenas por `root`.

Os principais arquivos relacionados a contas de usuário, atributos e controle de acesso são:

`/etc/passwd`

Este arquivo armazena informações básicas sobre os usuários no sistema, incluindo UID e GID, diretório inicial, shell etc. Apesar do nome, nenhuma senha é armazenada aqui.

`/etc/group`

Este arquivo armazena informações básicas sobre todos os grupos de usuários no sistema, como nome do grupo, GID e membros.

`/etc/shadow`

É aqui que as senhas dos usuários são armazenadas. Elas são criptografadas por segurança.

`/etc/gshadow`

Este arquivo armazena informações mais detalhadas sobre grupos, incluindo uma senha criptografada com hash que permite que os usuários se tornem membros do grupo temporariamente, uma lista de usuários que podem se tornar membros do grupo a qualquer momento e uma lista de administradores do grupo.

WARNING Esses arquivos não foram projetados para ser editados diretamente e isso jamais deve ser feito. Esta lição aborda apenas as informações armazenadas nesses arquivos e não a edição deles.

Por padrão, qualquer usuário pode acessar o `/etc` e ler os arquivos `/etc/passwd` e `/etc/group`. E também por padrão nenhum usuário, exceto `root`, pode ler os arquivos `/etc/shadow` ou `/etc/gshadow`.

Também existem arquivos relacionados ao aumento básico de privilégios em sistemas Linux, como nos comandos `su` e `sudo`. Por padrão, eles só estão acessíveis ao usuário `root`.

`/etc/sudoers`

Este arquivo controla quem pode usar o comando `sudo` e como.

`/etc/sudoers.d`

Este diretório pode conter arquivos que suplementam as configurações do arquivo `sudoers`.

No que diz respeito ao exame LPI Linux Essentials, basta conhecer o caminho e o nome do arquivo padrão de configuração do sudo, `/etc/sudoers`. A configuração dele está além do escopo

destes materiais.

WARNING Embora `/etc/sudoers` seja um arquivo de texto, ele jamais deve ser editado diretamente. Se for necessário fazer alterações em seu conteúdo, elas devem ser efetuadas através do utilitário `visudo`.

/etc/passwd

O arquivo `/etc/passwd` é popularmente chamado "arquivo password" (senha). Cada linha contém diversos campos, sempre delimitados por dois pontos (`:`). Apesar do nome, o hash one-way das senhas não é armazenado neste arquivo atualmente.

A sintaxe típica de uma linha nesse arquivo é:

```
USERNAME:PASSWORD:UID:GID:GECOS:HOMEDIR:SHELL
```

Onde:

`USERNAME`
: O nome de usuário ou nome de login (nome), como `root`, `nobody`, `emma`.

`PASSWORD`
: Antigo local do hash de senhas. Quase sempre é `x`, indicando que a senha está armazenada no arquivo `/etc/shadow`.

`UID`
: ID de usuário (UID), como `0`, `99`, `1024`.

`GID`
: ID de grupo padrão (GID), como `0`, `99`, `1024`.

`GECOS`
: Uma lista em formato CSV de informações de usuários, incluindo nome, localização, número de telefone. Por exemplo: `Emma Smith,42 Douglas St,555.555.5555`

`HOMEDIR`
: Caminho para o diretório inicial do usuário, como `/root`, `/home/emma`, etc.

`SHELL`
: O shell padrão para este usuário, como `/bin/bash`, `/sbin/nologin`, `/bin/ksh` etc.

Por exemplo, a linha a seguir descreve o usuário `emma`:

```
emma:x:1000:1000:Emma Smith,42 Douglas St,555.555.5555:/home/emma:/bin/bash
```

Compreendendo o campo GECOS

O campo GECOS contém três (3) ou mais campos delimitados por vírgula (`,`), sendo também conhecido como lista de Valores Separados por Vírgula (*Comma Separated Values*, ou CSV). Embora não exista um padrão imposto, os campos geralmente estão na seguinte ordem:

```
NAME,LOCATION,CONTACT
```

Onde:

`NAME`
: é o “Full Name” (nome completo) do usuário, ou o “Software Name” (nome do software) no caso de uma conta de serviço.

`LOCATION`
: normalmente é a localização física do usuário dentro de um prédio, o número da sala ou o departamento ou pessoa a contatar no caso de uma conta de serviço.

`CONTACT`
: lista as informações de contato, como o telefone pessoal ou profissional.

Os campos adicionais podem incluir outras informações de contato, como um número residencial ou endereço de email. Para alterar as informações no campo GECOS, use o comando `chfn` e responda às perguntas, como abaixo. Se nenhum nome de usuário for fornecido após o nome do comando, serão alteradas as informações do usuário atual:

```
$ chfn

Changing the user information for emma
Enter the new value, or press ENTER for the default
    Full Name: Emma Smith
    Room Number []: 42
    Work Phone []: 555.555.5555
    Home Phone []: 555.555.6666
```

/etc/group

O arquivo `/etc/group` contém arquivos delimitados por dois pontos (`:`), armazenando informações básicas sobre os grupos no sistema. Ele às vezes é chamado “arquivo de grupo”. A sintaxe de cada linha é:

```
NAME:PASSWORD:GID:MEMBERS
```

Onde:

`NAME`
: é o nome do grupo, como `root`, `users`, `emma` etc.

`PASSWORD`
: Antigo local do hash opcional de senhas de grupos. Quase sempre é `x`, indicando que a senha (caso definida) está armazenada no arquivo `/etc/gshadow`.

`GID`
: ID de grupo (GID), como `0`, `99`, `1024`.

`MEMBERS`
: uma lista de nomes de usuário que são membros do grupo, separados por vírgula, como `jsmith,emma`.

O exemplo abaixo mostra uma linha contendo informações sobre o grupo `students`:

```
students:x:1023:jsmith,emma
```

O usuário não precisa ser listado no campo members quando o grupo é o grupo principal de um usuário. Se um usuário estiver listado, o item é redundante — ou seja, não há alterações na funcionalidade, estando listado ou não.

NOTE O uso de senhas para grupos está além do escopo desta seção; no entanto, se houver uma, o hash da senha é armazenado no arquivo `/etc/gshadow`. Isso também está além do escopo desta seção.

/etc/shadow

A tabela a seguir lista os atributos armazenados no arquivo `/etc/shadow`, comumente chamado de “arquivo de sombra”. O arquivo contém campos sempre delimitados por dois pontos (`:`).

Embora o arquivo tenha muitos campos, a maioria está além do escopo desta lição, exceto os dois primeiros.

A sintaxe básica para uma linha nesse arquivo é:

```
USERNAME:PASSWORD:LASTCHANGE:MINAGE:MAXAGE:WARN:INACTIVE:EXPDATE
```

Onde:

`USERNAME`

O nome de usuário (igual a `/etc/passwd`), como `root`, `nobody`, `emma`.

`PASSWORD`

Um hash unidirecional da senha, incluindo o sal anterior. Por exemplo: `!!`, `!$1$01234567$ABC...`, `$6$012345789ABCDEF$012...`.

`LASTCHANGE`

Data da última alteração da senha em dias desde a "época", como `17909`.

`MINAGE`

Idade mínima da senha em dias.

`MAXAGE`

Idade máxima da senha em dias.

`WARN`

Período de aviso antes da expiração da senha, em dias.

`INACTIVE`

Idade máxima da senha após a expiração, em dias.

`EXPDATE`

Data da expiração da senha, em dias desde a "época".

No exemplo abaixo vemos um exemplo de entrada do arquivo `/etc/shadow`. Note que certos valores, como `INACTIVE` e `EXPDATE`, estão indefinidos.

```
emma:$6$nP532JDDogQYZF8I$bjFNh9eT1xpb9/n6pmjlIwgu7hGjH/eytSdttbmVv0MlyTMFgBIXESFNUmTo9EGxxH1
OT1HGQzR0so4n1npbE0:18064:0:99999:7:::
```

A “época” de um sistema POSIX é meia-noite (0000) do Tempo Universal Coordenado (UTC), na quinta-feira, 1º de janeiro de 1970. A maioria das datas e horas POSIX estão em segundos desde a “época” ou, no caso do arquivo `/etc/shadow`, dias desde a “época”.

NOTE O arquivo de sombra foi desenvolvido para ser legível apenas pelo superusuário e serviços selecionados de autenticação do sistema principal que verificam o hash de senha unidirecional no login ou em outro momento da autenticação.

Embora existam soluções de autenticação diferentes, o método básico de armazenamento de senha é a *função de hash* unidirecional (one-way). Isso é feito para que a senha nunca seja armazenada em texto não criptografado em um sistema, já que a função de hash não é reversível. Você pode transformar uma senha em um hash, mas (idealmente) não é possível transformar um hash novamente em uma senha.

No máximo, é necessário um método de força bruta para experimentar todas as combinações de uma senha até que uma delas corresponda. Para atenuar o risco de um hash de senha ser quebrado em um sistema, os sistemas Linux aplicam um “sal” aleatório em cada hash de senha de um usuário. Assim, o hash para uma senha de usuário em um sistema Linux geralmente não será o mesmo que em outro sistema Linux, mesmo que a senha seja idêntica.

No arquivo `/etc/shadow`, a senha pode assumir diversas formas. Elas tipicamente incluem o seguinte:

`!!`

Indica uma conta “desativada” (a autenticação não é possível), sem hash de senha armazenado.

`!$1$01234567$ABC…`

Uma conta “desativada” (indicado pelo ponto de exclamação inicial), com uma função de hash anterior, sal de hash e string de hash armazenados.

`$1$0123456789ABC$012…`

Uma conta “ativada”, com uma função de hash, sal de hash e string de hash armazenados.

A função de hash, o sal de hash e a string de hash são precedidos e delimitados por um cifrão (`$`). O comprimento do sal de hash deve ser entre oito e dezesseis (8-16) caracteres. Eis exemplos dos três mais comuns:

`$1$01234567$ABC…`

Uma função de hash de MD5 (`1`), com um hash de exemplo com oito caracteres.

`$5$01234567ABCD$012...`

Uma função de hash de SHA256 (`5`), com um hash de exemplo com doze caracteres.

`$6$01234567ABCD$012...`

Uma função de hash de SHA512 (`6`), com um hash de exemplo com doze caracteres.

NOTE

A função de hash MD5 é considerada criptograficamente insegura em relação ao nível atual (2010 e posterior) dos ASICs e até mesmo para os SIMD usados para computação em geral. Na verdade, o FIPS (Federal Information Processing Standards) dos EUA não permite que o MD5 seja usado para quaisquer funções criptográficas, exceto em aspectos muito limitados de validação, mas não para garantir a integridade das assinaturas digitais ou finalidades similares.

Do ponto de vista dos objetivos e do exame LPI Linux Essentials, basta entender que o hash da senha de um usuário local é armazenado no arquivo `/etc/shadow`, que apenas serviços selecionados de autenticação podem ler e o superusuário pode modificar através de outros comandos.

Exercícios Guiados

1. Considere a seguinte saída do comando `id`:

```
$ id emma
uid=1000(emma) gid=1000(emma)
groups=1000(emma),4(adm),5(tty),10(uucp),20(dialout),27(sudo),46(plugdev)
```

   Em quais arquivos estão armazenados os atributos a seguir?

UID e GID	
Grupos	

   - Adicionalmente, em qual arquivo a senha do usuário é armazenada?

2. Quais dos tipos de criptografia a seguir é usado por padrão para armazenar senhas localmente em um sistema Linux?
   - Asymmetric
   - One-way Hash
   - Symmetric
   - ROT13

3. Se uma conta tem um ID de usuário (UID) com número menor que 1000, de que tipo de conta se trata?

4. Como é possivel obter uma lista dos logins ativos de seu sistema e uma contagem deles?

5. Usando o comando `grep`, temos o resultado abaixo com informações sobre o usuário `emma`.

```
$ grep emma /etc/passwd
emma:x:1000:1000:Emma Smith,42 Douglas St,555.555.5555,:/home/emma:/bin/ksh
```

   Preencha as lacunas da tabela com as informações apropriadas usando a saída do comando

anterior.

Nome de usuário	
Senha	
UID	
GID principal	
GECOS	
Diretório inicial	
Shell	

Exercícios Exploratórios

1. Compare os resultados de `last` com `w` e `who`. Quais detalhes estão faltando em cada um dos comandos em comparação com os outros?

2. Tente lançar os comandos `who` e `w -his`.
   - Quais informações foram removidas da saída do comando `w` com as opções “no header” (`-h`) e “short” (`-s`)?
   - Quais informações foram adicionadas à saída do comando `w` com a opção “ip address” (`-i`)?

3. Qual é o arquivo que armazena o hash de senha unidirecional de uma conta de usuário?

4. Qual arquivo contém a lista dos grupos de que uma conta de usuário faz parte? Qual lógica poderia ser usada para compilar uma lista dos grupos de que uma conta de usuário faz parte?

5. Um ou mais (1+) dos seguintes arquivos não é legível por usuários regulares, sem privilégios especiais, por padrão. Quais?
   - `/etc/group`
   - `/etc/passwd`
   - `/etc/shadow`
   - `/etc/sudoers`

6. Como seria possível trocar o shell de login do usuário atual pelo Korn Shell (`/usr/bin/ksh`) em modo não-interativo?

7. Por que o diretório inicial do usuário `root` não fica dentro do diretório `/home`?

Resumo

Nesta lição, descobrimos os bancos de dados de usuários e grupos do Linux. Aprendemos as propriedades mais importantes de usuários e grupos, incluindo seus nomes e IDs numéricos. Também investigamos como o hash de senhas funciona no Linux e como os grupos são atribuídos aos usuários.

Todas essas informações são armazenadas nos quatro arquivos a seguir, que fornecem os controles de acesso de segurança local mais básicos em um sistema Linux:

`/etc/passwd`
: Todos os atributos POSIX locais da conta de usuário, exceto o hash da senha, legíveis por todos.

`/etc/group`
: Todos os atributos POSIX locais da conta de grupo, legíveis por todos.

`/etc/shadow`
: Todos os hashes de senha locais de usuário (e informações de expiração), ilegíveis por qualquer um (apenas processos selecionados).

`/etc/sudoers`
: Todas as informações/permissões locais de aumento de privilégios pelo comando `sudo`.

Os seguintes comandos foram discutidos nesta lição:

`id`
: Lista IDs de usuários e grupos reais (ou efetivos).

`last`
: Lista os usuários que fizeram login por último.

`who`
: Lista usuários que estão conectados no momento.

`w`
: Semelhante a `who`, mas com contexto adicional.

`su`
: Alterna para outro usuário com um shell de login ou executa comandos como aquele usuário, contornando a senha desse usuário.

sudo

Switch User Do (Substituição de Usuário) ou Superuser Do (Fazer como Superusuário). Caso esteja autorizado, o usuário atual digita sua própria senha (se necessário) para aumentar os privilégios.

chsh

Troca o shell de um usuário.

chfn

Altera as informações do usuário no campo GECOS.

Respostas aos Exercícios Guiados

1. Considere a seguinte saída do comando `id`:

   ```
   $ id emma
   uid=1000(emma) gid=1000(emma)
   groups=1000(emma),4(adm),5(tty),10(uucp),20(dialout),27(sudo),46(plugdev)
   ```

   Em quais arquivos estão armazenados os atributos a seguir?

UID e GID	`/etc/passwd`
Grupos	`/etc/group`

   - Adicionalmente, em qual arquivo a senha do usuário é armazenada?

     A senha de usuário criptografada fica armazenada em `/etc/shadow`.

2. Quais dos tipos de criptografia a seguir é usado por padrão para armazenar senhas localmente em um sistema Linux?

   Por padrão, um hash unidirecional é usado para armazenar senhas.

3. Se uma conta tem um ID de usuário (UID) com número menor que 1000, de que tipo de conta se trata?

   Contas com UID menor que 1000 geralmente são contas de sistema.

4. Como é possivel obter uma lista dos logins ativos de seu sistema e uma contagem deles?

   Use o comando `w`. Além de uma lista de logins ativos, ele também mostra informações como a quantidade de usuários conectados, além da carga e tempo de atividade do sistema.

5. Usando o comando `grep`, temos o resultado abaixo com informações sobre o usuário `emma`.

   ```
   $ grep emma /etc/passwd
   emma:x:1000:1000:Emma Smith,42 Douglas St,555.555.5555,:/home/emma:/bin/ksh
   ```

   Preencha as lacunas da tabela com as informações apropriadas usando a saída do comando anterior.

Nome de usuário	`emma`
Senha	`x` - deve sempre ser `x` para um login ativo válido
UID	`1000`
GID principal	`1000`
GECOS	`Emma Smith,42 Douglas St,555.555.5555`
Diretório inicial	`/home/emma`
Shell	`/bin/ksh`

Respostas aos Exercícios Exploratórios

1. Compare os resultados de `last` com `w` e `who`. Quais detalhes estão faltando em cada um dos comandos em comparação com os outros?

   As ferramentas `w` e `who` listam apenas os usuários atualmente conectados ao sistema, ao passo que `last` também lista os usuários que se desconectaram. O comando `w` lista a utilização do sistema, enquanto `who` não faz isso.

2. Tente lançar os comandos `who` e `w -his`.
   - Quais informações foram removidas da saída do comando `w` com as opções “no header” (`-h`) e “short” (`-s`)?

     O cabeçalho não é impresso, o que é útil para análise, e o tempo de login e informações selecionadas da CPU não são listados, respectivamente.

   - Quais informações foram adicionadas à saída do comando `w` com a opção “ip address” (`-i`)?

     Essa opção imprime o endereço IP, em vez de tentar a resolução de DNS, imprimindo o nome do host. Essa opção de `w` é mais adequada à saída padrão do comando `last`.

3. Qual é o arquivo que armazena o hash de senha unidirecional de uma conta de usuário?

   O arquivo `/etc/shadow` armazena o hash unidirecional de uma conta de usuário, já que ele não é legível por uma conta de usuário regular, sem privilégios, ao contrário do arquivo `/etc/passwd`.

4. Qual arquivo contém a lista dos grupos de que uma conta de usuário faz parte? Qual lógica poderia ser usada para compilar uma lista dos grupos de que uma conta de usuário faz parte?

   O arquivo `/etc/group` tem uma lista em formato CSV de nomes de usuários no último campo (“members”) de qualquer linha correspondente a um grupo.

   Qualquer linha no arquivo `/etc/group` na qual o usuário aparece listado no último campo, “members”, implica que o usuário é membro daquele grupo — supondo-se que ele esteja corretamente formatado (valores delimitados por vírgulas). Além disso, a participação do usuário em seu grupo principal no arquivo `/etc/passwd` também terá uma entrada correspondente no arquivo `/etc/group`, incluindo tanto o nome do grupo quanto o GID.

5. Um ou mais (1+) dos seguintes arquivos não é legível por usuários regulares, sem privilégios especiais, por padrão. Quais?
   - `/etc/group`

- `/etc/passwd`
- `/etc/shadow`
- `/etc/sudoers`

  Os arquivos `/etc/shadow` e `/etc/sudoers` não são legíveis por padrão, exceto por serviços selecionados ou o superusuário. Estas respostas serão personalizadas com base nos sistemas e nomes de usuário usados no laboratório.

6. Como seria possível trocar o shell de login do usuário atual pelo Korn Shell (`/usr/bin/ksh`) em modo não-interativo?

```
$ chsh -s /usr/bin/ksh
```

7. Por que o diretório inicial do usuário `root` não fica dentro do diretório `/home`?

   Porque a conta `root` é necessária para resolver problemas e corrigir erros, o que pode incluir problemas no sistema de arquivos relacionados ao diretório `/home`. Nesses casos, `root` deve estar plenamente funcional mesmo que o sistema de arquivos de `/home` esteja indisponível.

5.2 Criando Usuários e Grupos

Referência ao LPI objectivo

Linux Essentials version 1.6, Exam 010, Objective 5.2

Peso

2

Áreas chave de conhecimento

- Comandos de usuários e de grupos
- IDs de usuários

Segue uma lista parcial de arquivos, termos e utilitários utilizados

- `/etc/passwd`, `/etc/shadow`, `/etc/group`, `/etc/skel/`
- `useradd`, `groupadd`
- `passwd`

5.2 Lição 1

Certificação:	Linux Essentials
Versão:	1.6
Tópico:	5 Segurança e permissões de arquivos
Objetivo:	5.2 Criação de usuários e grupos
Lição:	1 de 1

Introdução

Gerenciar usuários e grupos em uma máquina Linux é um dos principais aspectos da administração do sistema. De fato, o Linux é um sistema operacional multiusuário no qual vários usuários podem usar a mesma máquina ao mesmo tempo.

As informações sobre usuários e grupos são armazenadas em quatro arquivos na árvore de diretórios `/etc/`:

`/etc/passwd`
: um arquivo de sete campos delimitados por dois pontos contendo informações básicas sobre os usuários

`/etc/group`
: um arquivo de quatro campos delimitados por dois pontos contendo informações básicas sobre os grupos

`/etc/shadow`
: um arquivo de nove campos delimitados por dois pontos contendo senhas de usuário

criptografadas

`/etc/gshadow`

um arquivo de quatro campos delimitados por dois pontos contendo senhas de grupo criptografadas

Todos esses arquivos são atualizados por um conjunto de ferramentas de linha de comando para gerenciamento de usuários e grupos, de que falaremos mais adiante nesta lição. Também existem aplicativos gráficos, específicos para cada distribuição Linux, com interfaces de gerenciamento mais simples e imediatas.

WARNING Mesmo que os arquivos sejam em texto simples, não os edite diretamente. Sempre use as ferramentas fornecidas com sua distribuição.

O Arquivo `/etc/passwd`

`/etc/passwd` é um arquivo legível por todos contendo uma lista de usuários em linhas separadas:

```
frank:x:1001:1001::/home/frank:/bin/bash
```

Cada linha consiste em sete campos delimitados por dois pontos:

Nome de usuário

O nome usado quando o usuário se loga no sistema.

Senha

A senha criptografada (ou um `x` no caso de senhas shadow).

ID de usuário (UID)

O número de identificação atribuído ao usuário no sistema.

ID de grupo (GID)

O número do grupo principal do usuário no sistema.

GECOS

Um campo de comentário opcional, usado para adicionar informações extras sobre o usuário (como o nome completo). O campo pode conter diversas entradas separadas por vírgula.

Diretório inicial

O caminho absoluto do diretório inicial do usuário.

Shell

O caminho absoluto do programa que é iniciado automaticamente quando o usuário efetua login no sistema (geralmente um shell interativo como `/bin/bash`).

O arquivo `/etc/group`

`/etc/group` é um arquivo legível por todos contendo uma lista de grupos em linhas separadas:

```
developer:x:1002:
```

Cada linha consiste em quatro campos delimitados por dois pontos:

Nome do grupo

O nome do grupo.

Senha do grupo

A senha criptografada do grupo (ou um `x` se forem usadas senhas shadow).

ID do grupo (GID)

O número de identificação atribuído ao grupo no sistema.

Lista de membros

Uma lista delimitada por vírgulas de usuários pertencentes ao grupo, exceto aqueles cujo grupo principal é este.

O arquivo `/etc/shadow`

`/etc/shadow` é um arquivo legível apenas pelo usuário root ou com privilégios de root e contém as senhas criptografadas dos usuários em linhas separadas:

```
frank:$6$i9gjM4Md4MuelZCd$7jJa8Cd2bbADFH4dwtfvTvJLOYCCCBf/.jYbK1IMYx7Wh4fErXcc2xQVU2N1gb97yI
YaiqH.jjJammzof2Jfr/:18029:0:99999:7:::
```

Cada linha consiste em nove campos delimitados por dois pontos:

Nome de usuário

O nome usado quando o usuário se loga no sistema.

Senha criptografada

A senha do usuário criptografada (se o valor for `!`, a conta está bloqueada).

Data da última alteração de senha

A data da última mudança de senha em número de dias desde 01/01/1970. Um valor de `0` indica que o usuário precisa trocar a senha em seu próximo acesso.

Idade mínima da senha

O número mínimo de dias que o usuário deve aguardar após uma alteração de senha para poder trocar a senha novamente.

Idade máxima da senha

O número máximo de dias que devem passar antes que uma alteração de senha seja solicitada.

Período de aviso de senha

O número de dias para a expiração da senha, durante os quais o usuário é avisado de que a senha deve ser alterada.

Período de inatividade da senha

O número de dias após a expiração de uma senha, durante os quais o usuário deve atualizá-la. Após esse período, se o usuário não alterar a senha, a conta é desativada.

Data de expiração da conta

A data, em número de dias desde 01/01/1970, na qual a conta do usuário será desativada. Um campo vazio indica que a conta do usuário nunca expirará.

Um campo reservado

Um campo reservado para uso futuro.

O arquivo `/etc/gshadow`

`/etc/gshadow` é um arquivo legível apenas pelo root e por usuários com privilégios de root que contém senhas criptografadas para grupos em linhas separadas:

```
developer:$6$7QUIhUX1WdO6$H7kOYgsboLkDseFHpk04lwAtweSUQHipoxIgo83QNDxYtYwgmZTCU0qSCuCkErmyR2
63rvHiLctZVDR7Ya9Ai1::
```

Cada linha consiste em quatro campos delimitados por dois pontos:

Nome do grupo

O nome do grupo.

Senha criptografada

A senha criptografada do grupo (é usada quando um usuário que não é membro do grupo deseja ingressar no grupo usando o comando `newgrp` — se a senha começar com `!`, ninguém tem permissão de acessar o grupo com `newgrp`).

Administradores do grupo

Uma lista dos administradores do grupo delimitada por vírgulas (eles podem alterar a senha do grupo, bem como adicionar ou remover membros do grupo com o comando `gpasswd`)

Membros do grupo

Uma lista dos membros do grupo delimitada por vírgulas.

Agora que vimos onde as informações de usuários e grupos são armazenadas, vamos falar sobre as ferramentas de linha de comando mais importantes para atualizar esses arquivos.

Adicionando e removendo contas de usuário

No Linux, adicionamos uma nova conta de usuário com o comando `useradd` e excluímos uma conta de usuário com o comando `userdel`.

Se quiséssemos criar uma nova conta de usuário chamada `frank` com uma configuração padrão, executaríamos o seguinte:

```
# useradd frank
```

Após criar o novo usuário, definimos uma senha usando `passwd`:

```
# passwd frank
Changing password for user frank.
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
```

Ambos os comandos requerem autoridade de root. Quando executamos o comando `useradd`, as informações de usuário e grupo armazenadas nos bancos de dados de senha e de grupo são atualizadas para a conta de usuário recém-criada e, caso especificado, o diretório inicial do novo usuário é criado, bem como um grupo com o mesmo nome da conta de usuário.

TIP

Lembre-se de que sempre podemos usar o utilitário `grep` para filtrar os bancos de dados de senhas e grupos, exibindo apenas a entrada referente a um usuário ou grupo específico. Para o exemplo acima, usaríamos

```
cat /etc/passwd | grep frank
```

ou

```
grep frank /etc/passwd
```

para ver informações básicas sobre a conta `frank` recém-criada.

As opções mais importantes que se aplicam ao comando `useradd` são:

`-c`

Cria uma nova conta de usuário com comentários personalizados (por exemplo, nome completo)

`-d`

Cria uma nova conta de usuário com um diretório inicial personalizado.

`-e`

Cria uma nova conta de usuário com uma data específica na qual ela será desativada.

`-f`

Cria uma nova conta de usuário definindo o número de dias que o usuário tem para atualizar a senha após a expiração.

`-g`

Cria uma nova conta de usuário com um GID específico.

`-G`

Cria uma nova conta de usuário adicionando-a a diversos grupos secundários.

`-m`

Cria uma nova conta de usuário com seu diretório inicial.

`-M`

Cria uma nova conta de usuário sem seu diretório inicial.

`-s`

Cria uma nova conta de usuário com um shell de login específico.

`-u`

Cria uma nova conta de usuário com um UID específico.

Depois que a nova conta de usuário é criada, usamos os comandos `id` e `groups` para descobrir seu UID, GID e os grupos aos quais pertence.

```
# id frank
uid=1000(frank) gid=1000(frank) groups=1000(frank)
# groups frank
frank : frank
```

TIP

Lembre-se de verificar e, possivelmente, editar o arquivo `/etc/login.defs`, que define os parâmetros de configuração que controlam a criação de usuários e grupos. Por exemplo, você pode definir o intervalo de UIDs e GIDs que podem ser atribuídos a novas contas de usuário e grupo, especificar que não é necessário usar a opção `-m` para criar o diretório inicial do novo usuário e se o sistema deve criar automaticamente um novo grupo para cada novo usuário.

Se quiser excluir uma conta de usuário, pode usar o comando `userdel`. Esse comando atualiza, em particular, as informações armazenadas nos bancos de dados da conta, excluindo todas as entradas referentes ao usuário especificado. A opção `-r` também remove o diretório inicial do usuário e todo o seu conteúdo, juntamente com o spool de correio do usuário. Os arquivos que estiverem em outros locais deverão ser pesquisados e excluídos manualmente.

```
# userdel -r frank
```

Também neste caso, é preciso ter autoridade root para excluir contas de usuário.

O diretório de esqueleto

Quando adicionamos uma nova conta de usuário e criamos seu diretório pessoal, este é preenchido com arquivos e pastas copiados do diretório de esqueleto (por padrão, `/etc/skel`). A idéia é simples: um administrador de sistema quer que os novos usuários tenham os mesmos arquivos e diretórios em seu diretório inicial. Portanto, se você deseja personalizar os arquivos e pastas que são criados automaticamente no diretório inicial das novas contas de usuário, adicione esses novos arquivos e pastas ao diretório de esqueleto.

TIP

Note que os arquivos de perfil geralmente encontrados no diretório de esqueleto são arquivos ocultos. Portanto, se você deseja listar todos os arquivos e pastas do diretório de esqueleto que serão copiados para o diretório inicial dos usuários recém-

> criados, deve usar o comando `ls -Al`.

Adicionando e excluindo grupos

Quanto ao gerenciamento dos grupos, os comandos `groupadd` e `groupdel` servem para adicionar e excluir grupos.

Para criar um novo grupo chamado `developer`, executaríamos o seguinte comando como root:

```
# groupadd -g 1090 developer
```

A opção `-g` deste comando cria um grupo com um GID específico.

Se quiser remover o grupo `developer`, você pode executar este comando:

```
# groupdel developer
```

WARNING Lembre-se de que quando adicionamos uma nova conta de usuário, o grupo principal e os grupos secundários aos quais ela pertence devem existir antes de se iniciar o comando `useradd`. Além disso, não é possível excluir um grupo se este for o grupo principal de uma conta de usuário.

O comando `passwd`

Este comando é usado principalmente para alterar a senha de um usuário. Qualquer usuário pode alterar sua senha, mas apenas o root pode alterar a senha de qualquer usuário.

Dependendo da opção de `passwd` usada, podemos controlar aspectos específicos do envelhecimento da senha:

`-d`
: Exclui a senha de uma conta de usuário (definindo assim uma senha vazia, transformando-a em uma conta sem senha).

`-e`
: Força a conta de usuário a alterar a senha.

`-l`
: Bloqueia a conta de usuário (a senha criptografada é prefixada com um ponto de exclamação).

`-u`

Desbloqueia a conta de usuário (o ponto de exclamação é removido).

`-S`

Exibe informações sobre o status da senha de uma conta específica.

Essas opções só estão disponíveis para o root. Para ver a lista completa de opções, consulte as páginas de manual.

Exercícios Guiados

1. Indique a que arquivo se refere cada uma das entradas a seguir:
   - `developer:x:1010:frank,grace,dave`
   - `root:x:0:0:root:/root:/bin/bash`
   - `henry:$1$.AbCdEfGh123456789A1b2C3d4.:18015:20:90:5:30::`
   - `henry:x:1000:1000:User Henry:/home/henry:/bin/bash`
   - `staff:!:dave:carol,emma`

2. Observe esta saída para responder às sete questões a seguir:

```
# cat /etc/passwd | tail -3
dave:x:1050:1050:User Dave:/home/dave:/bin/bash
carol:x:1051:1015:User Carol:/home/carol:/bin/sh
henry:x:1052:1005:User Henry:/home/henry:/bin/tcsh
# cat /etc/group | tail -3
web_admin:x:1005:frank,emma
web_developer:x:1010:grace,kevin,christian
dave:x:1050:
# cat /etc/shadow | tail -3
dave:$6$AbCdEfGh123456789A1b2C3D4e5F6G7h8i9:0:20:90:7:30::
carol:$6$q1w2e3r4t5y6u7i8AbcDeFgHiLmNoPqRsTu:18015:0:60:7:::
henry:!$6$123456789aBcDeFgHa1B2c3d4E5f6g7H8I9:18015:0:20:5:::
# cat /etc/gshadow | tail -3
web_admin:!:frank:frank,emma
web_developer:!:kevin:grace,kevin,christian
dave:!::
```

   - Qual o identificador de usuário (UID) e identificador de grupo (GID) de `carol`?
   - Qual shell está configurado para `dave` e `henry`?

- Qual o nome do grupo principal de `henry`?
- Quem são os membros do grupo `web_developer`? Quais deles são administradores do grupo?
- Qual usuário não pode se logar no sistema?
- Qual usuário deverá mudar a senha na próxima vez em que fizer login no sistema?
- Quantos dias devem se passar até que seja exigida uma alteração de senha para `carol`?

Exercícios Exploratórios

1. Trabalhando como root, use o comando `useradd -m dave` para adicionar uma nova conta de usuário. Quais as operações realizadas por esse comando? Suponha que `CREATE_HOME` e `USERGROUPS_ENAB` em `/etc/login.defs` estejam definidos como sim.

2. A conta de usuário `dave` foi criada; esse usuário pode fazer login no sistema?

3. Encontre o identificador de usuário (UID) e de grupo (GID) de `dave` e todos os membros do grupo `dave`.

4. Crie os grupos `sys_admin`, `web_admin` e `db_admin` e encontre seus identificadores de grupo (GIDs).

5. Adicione uma nova conta de usuário de nome `carol` com UID 1035 e defina `sys_admin` como grupo principal e `web_admin` e `db_admin` como grupos secundários.

6. Exclua as contas de usuário `dave` e `carol` e os grupos `sys_admin`, `web_admin` e `db_admin` criados anteriormente.

7. Execute o comando `ls -l /etc/passwd /etc/group /etc/shadow /etc/gshadow` e descreva a saída em termos de permissões de arquivos. Quais desses quatro arquivos estão em shadow por razões de segurança? Pressuponha que seu sistema usa senhas shadow.

8. Execute o comando `ls -l /usr/bin/passwd`. Qual bit especial é definido e o que ele significa?

Resumo

Nesta lição, você aprendeu:

- Noções fundamentais de gerenciamento de usuários e grupos no Linux
- Gerenciar informações de usuários e grupos armazenados nos bancos de dados de senhas e grupos
- Manter o diretório de esqueleto
- Adicionar e remover contas de usuário
- Adicionar e remover contas de grupo
- Alterar a senha das contas de usuário

Os seguintes comandos foram discutidos nesta lição:

`useradd`
: Cria uma nova conta de usuário.

`groupadd`
: Cria uma nova conta de grupo.

`userdel`
: Exclui uma conta de usuário.

`groupdel`
: Exclui uma conta de grupo.

`passwd`
: Altera a senha das contas de usuário e controla todos os aspectos do envelhecimento da senha.

Respostas aos Exercícios Guiados

1. Indique a que arquivo se refere cada uma das entradas a seguir:
   - `developer:x:1010:frank,grace,dave`

     `/etc/group`
   - `root:x:0:0:root:/root:/bin/bash`

     `/etc/passwd`
   - `henry:$1$.AbCdEfGh123456789A1b2C3d4.:18015:20:90:5:30::`

     `/etc/shadow`
   - `henry:x:1000:1000:User Henry:/home/henry:/bin/bash`

     `/etc/passwd`
   - `staff:!:dave:carol,emma`

     `/etc/gshadow`
2. Observe esta saída para responder às sete questões a seguir:

```
# cat /etc/passwd | tail -3
dave:x:1050:1050:User Dave:/home/dave:/bin/bash
carol:x:1051:1015:User Carol:/home/carol:/bin/sh
henry:x:1052:1005:User Henry:/home/henry:/bin/tcsh
# cat /etc/group | tail -3
web_admin:x:1005:frank,emma
web_developer:x:1010:grace,kevin,christian
dave:x:1050:
# cat /etc/shadow | tail -3
dave:$6$AbCdEfGh123456789A1b2C3D4e5F6G7h8i9:0:20:90:7:30::
carol:$6$q1w2e3r4t5y6u7i8AbcDeFgHiLmNoPqRsTu:18015:0:60:7:::
henry:!$6$123456789aBcDeFgHa1B2c3d4E5f6g7H8I9:18015:0:20:5:::
# cat /etc/gshadow | tail -3
web_admin:!:frank:frank,emma
web_developer:!:kevin:grace,kevin,christian
dave:!::
```

   - Qual o identificador de usuário (UID) e identificador de grupo (GID) de `carol`?

O identificador de usuário é 1051 e o de grupo é 1015 (o terceiro e o quarto campos de `/etc/passwd`).

- Qual shell está configurado para `dave` e `henry`?

  `dave` usa /bin/bash e `henry` usa /bin/tcsh (o sétimo campo em `/etc/passwd`).

- Qual o nome do grupo principal de `henry`?

  O nome do grupo é `web_admin` (o primeiro campo em `/etc/group`).

- Quem são os membros do grupo `web_developer`? Quais deles são administradores do grupo?

  Os membros são `grace`, `kevin` e `christian` (o quarto campo em `/etc/group`), mas apenas `kevin` é o administrador do grupo (o terceiro campo em `/etc/gshadow`).

- Qual usuário não pode se logar no sistema?

  A conta de usuário `henry` está bloqueada (tem um ponto de exclamação na frente dos hashes de senha em `/etc/shadow`).

- Qual usuário deverá mudar a senha na próxima vez em que fizer login no sistema?

  Se o terceiro campo (Data da Última Mudança de Senha) de `/etc/shadow` for 0, o usuário deve alterar sua senha na próxima vez em que se logar ao sistema. Portanto, `dave` deverá trocar a senha.

- Quantos dias devem se passar até que seja exigida uma alteração de senha para `carol`?

  60 dias (o quinto campo em `/etc/shadow`).

Respostas aos Exercícios Exploratórios

1. Trabalhando como root, use o comando `useradd -m dave` para adicionar uma nova conta de usuário. Quais as operações realizadas por esse comando? Suponha que `CREATE_HOME` e `USERGROUPS_ENAB` em `/etc/login.defs` estejam definidos como sim.

   O comando adiciona um novo usuário, chamado `dave`, à lista de usuários no sistema. O diretório inicial de `dave` é criado (por padrão `/home/dave`) e os arquivos e diretórios contidos no diretório esqueleto são copiados no diretório inicial. Finalmente, o novo grupo é criado com o mesmo nome da conta de usuário.

2. A conta de usuário `dave` foi criada; esse usuário pode fazer login no sistema?

   Não, porque a conta `dave` está bloqueada (como indica o ponto de exclamação em `/etc/shadow`).

   ```
   # cat /etc/shadow | grep dave
   dave:!:18015:0:99999:7:::
   ```

   Se você definir uma senha para `dave`, a conta será desbloquada. Faça isso usando o comando `passwd`.

   ```
   # passwd dave
   Changing password for user dave.
   New UNIX password:
   Retype new UNIX password:
   passwd: all authentication tokens updated successfully.
   ```

3. Encontre o identificador de usuário (UID) e de grupo (GID) de `dave` e todos os membros do grupo `dave`.

   ```
   # cat /etc/passwd | grep dave
   dave:x:1015:1019::/home/dave:/bin/sh
   # cat /etc/group | grep 1019
   dave:x:1019:
   ```

   O UID e GID of `dave` são 1015 e 1019, respectivamente (o terceiro e o quarto campos em `/etc/passwd`) e o grupo `dave` não tem membros (o quarto campo em `/etc/group` está vazio).

4. Crie os grupos `sys_admin`, `web_admin` e `db_admin` e encontre seus identificadores de grupo

(GIDs).

```
# groupadd sys_admin
# groupadd web_admin
# groupadd db_admin
# cat /etc/group | grep admin
sys_admin:x:1020:
web_admin:x:1021:
db_admin:x:1022:
```

Os GIDs dos grupos `sys_admin`, `web_admin` e `db_admin` são 1020, 1021 e 1022, respectivamente.

5. Adicione uma nova conta de usuário de nome `carol` com UID 1035 e defina `sys_admin` como grupo principal e `web_admin` e `db_admin` como grupos secundários.

```
# useradd -u 1035 -g 1020 -G web_admin,db_admin carol
# id carol
uid=1035(carol) gid=1020(sys_admin) groups=1020(sys_admin),1021(web_admin),1022(db_admin)
```

6. Exclua as contas de usuário `dave` e `carol` e os grupos `sys_admin`, `web_admin` e `db_admin` criados anteriormente.

```
# userdel -r dave
# userdel -r carol
# groupdel sys_admin
# groupdel web_admin
# groupdel db_admin
```

7. Execute o comando `ls -l /etc/passwd /etc/group /etc/shadow /etc/gshadow` e descreva a saída em termos de permissões de arquivos. Quais desses quatro arquivos estão em shadow por razões de segurança? Pressuponha que seu sistema usa senhas shadow.

```
# ls -l /etc/passwd /etc/group /etc/shadow /etc/gshadow
-rw-r--r-- 1 root root    853 mag  1 08:00 /etc/group
-rw-r----- 1 root shadow 1203 mag  1 08:00 /etc/gshadow
-rw-r--r-- 1 root root   1354 mag  1 08:00 /etc/passwd
-rw-r----- 1 root shadow 1563 mag  1 08:00 /etc/shadow
```

Os arquivos `/etc/passwd` e `/etc/group` são legíveis por todos e estão em shadow por motivos

de segurança. Quando senhas shadow são usadas, vemos um `x` no segundo campo desses arquivos, porque as senhas criptografadas de usuários e grupos são armazenadas em `/etc/shadow` e `/etc/gshadow`, legíveis apenas pelo root e, em certos sistemas, também por membros pertencentes ao grupo `shadow`.

8. Execute o comando `ls -l /usr/bin/passwd`. Qual bit especial é definido e o que ele significa?

```
# ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 42096 mag 17  2015 /usr/bin/passwd
```

O comando `passwd` tem o bit SUID definido (o quarto caractere desta linha), o que significa que o comando é executado com os privilégios do proprietário do arquivo (ou seja, o root). É assim que os usuários comuns podem mudar sua senha.

5.3 Gerenciando permissões e donos de arquivos

Referência ao LPI objectivo

Linux Essentials version 1.6, Exam 010, Objective 5.3

Peso

2

Áreas chave de conhecimento

- Permissões de arquivos/diretórios e seus donos

Partial list of the used files, terms and utilities

- `ls -l`, `ls -a`
- `chmod`, `chown`

5.3 Lição 1

Certificação:	Linux Essentials
Versão:	1.6
Tópico:	5 Segurança e permissões de arquivos
Objetivo:	5.3 Gerenciamento de permissões e propriedade de arquivos
Lição:	1 de 1

Introdução

Por ser um sistema multiusuário, o Linux precisa de uma maneira de rastrear quem é o dono de cada arquivo e se um usuário pode ou não executar ações nesse arquivo. Isso serve para garantir a privacidade dos usuários que desejam manter em sigilo o conteúdo de seus arquivos, além de permitir trabalhar em colaboração, tornando certos arquivos acessíveis a diversos usuários.

Isso é feito por meio de um sistema de permissões em três níveis: cada arquivo no disco pertence a um usuário e a um grupo de usuários e conta com três conjuntos de permissões: uma para o proprietário, uma para o grupo que possui o arquivo e outra para todos os demais. Nesta lição, você aprenderá a consultar as permissões para um arquivo e a manipulá-las.

Como consultar informações sobre arquivos e diretórios

O comando `ls` é usado para obter uma lista do conteúdo de qualquer diretório. Nessa forma básica, aparecem somente os nomes de arquivos:

```
$ ls
```

```
Another_Directory  picture.jpg  text.txt
```

Mas há muito mais informações disponíveis sobre cada arquivo, incluindo tipo, tamanho, dono e muito mais. Para visualizar essas informações, você deve solicitar a `ls` uma lista de "forma longa", usando o parâmetro `-l`:

```
$ ls -l
total 536
drwxrwxr-x 2 carol carol   4096 Dec 10 15:57 Another_Directory
-rw------- 1 carol carol 539663 Dec 10 10:43 picture.jpg
-rw-rw-r-- 1 carol carol   1881 Dec 10 15:57 text.txt
```

Cada coluna na saída acima tem um significado:

- A *primeira* coluna da lista mostra o tipo de arquivo e as permissões.

  Por exemplo, em `drwxrwxr-x`:

  - O primeiro caractere, `d`, indica o tipo de arquivo.
  - Os três caracteres seguintes, `rwx`, indicam as permissões do proprietário do arquivo, também designado como *user* ou `u`.
  - Os três caracteres seguintes, `rwx`, indicam as permissões do *grupo* que possui o arquivo, também chamado de `g`.
  - Os três últimos caracteres, `r-x`, indicam as permissões para todos os outros, também chamados de *others* ou `o`.
- A *segunda* coluna indica o número de links fisicos apontando para aquele arquivo. No caso de um diretório, isso significa o número de subdiretórios, além de um link para si mesmo (`.`) e o diretório pai (`..`).
- A *terceira* e *quarta* colunas mostram as informações de propriedade: respectivamente o usuário e o grupo que possuem o arquivo.
- A *quinta* coluna mostra o tamanho do arquivo em bytes.
- A *sexta* coluna mostra a data e a hora exatas, ou a *marca temporal*, de quando o arquivo foi modificado pela última vez.
- A *sétima* e última coluna mostra o nome do arquivo.

Se quiser ver os tamanhos de arquivo em formato "legível por humanos", adicione o parâmetro `-h` a `ls`. O tamanho dos arquivos com menos de um quilobyte será mostrado em bytes. Nos arquivos com mais de um quilobyte e menos de um megabyte, um `K` é adicionado depois do

número, indicando que o tamanho está em quilobytes. O mesmo vale para arquivos com tamanhos em megabytes (`M`) gigabytes (`G`):

```
$ ls -lh
total 1,2G
drwxrwxr-x 2 carol carol 4,0K Dec 10 17:59 Another_Directory
----r--r-- 1 carol carol    0 Dec 11 10:55 foo.bar
-rw-rw-r-- 1 carol carol 1,2G Dec 20 18:22 HugeFile.zip
-rw------- 1 carol carol 528K Dec 10 10:43 picture.jpg
---xr-xr-x 1 carol carol   33 Dec 11 10:36 test.sh
-rwxr--r-- 1 carol carol 1,9K Dec 20 18:13 text.txt
-rw-rw-r-- 1 carol carol 2,6M Dec 11 22:14 Zipped.zip
```

Para exibir somente as informações de um conjunto específico de arquivos, adicione o nome desses arquivos a `ls`:

```
$ ls -lh HugeFile.zip test.sh
total 1,2G
-rw-rw-r-- 1 carol carol 1,2G Dec 20 18:22 HugeFile.zip
---xr-xr-x 1 carol carol   33 Dec 11 10:36 test.sh
```

E os diretórios?

Se você tentar consultar informações sobre um diretório usando `ls -l`, ele exibirá uma lista do conteúdo do diretório:

```
$ ls -l Another_Directory/
total 0
-rw-r--r-- 1 carol carol 0 Dec 10 17:59 another_file.txt
```

Para evitar isso e consultar informações sobre o diretório em si, adicione o parâmetro `-d` a `ls`:

```
$ ls -l -d Another_Directory/
drwxrwxr-x 2 carol carol 4096 Dec 10 17:59 Another_Directory/
```

Exibindo arquivos ocultos

A lista de diretórios que obtivemos usando `ls -l` anteriormente está incompleta:

```
$ ls -l
total 544
drwxrwxr-x 2 carol carol   4096 Dec 10 17:59 Another_Directory
-rw------- 1 carol carol 539663 Dec 10 10:43 picture.jpg
-rw-rw-r-- 1 carol carol   1881 Dec 10 15:57 text.txt
```

Há outros três arquivos nesse diretório, mas eles estão ocultos. No Linux, os arquivos cujo nome começa com um ponto (`.`) são ocultados automaticamente. Para vê-los, precisamos adicionar o parâmetro `-a` ao `ls`:

```
$ ls -l -a
total 544
drwxrwxr-x 3 carol carol   4096 Dec 10 16:01 .
drwxrwxr-x 4 carol carol   4096 Dec 10 15:56 ..
drwxrwxr-x 2 carol carol   4096 Dec 10 17:59 Another_Directory
-rw------- 1 carol carol 539663 Dec 10 10:43 picture.jpg
-rw-rw-r-- 1 carol carol   1881 Dec 10 15:57 text.txt
-rw-r--r-- 1 carol carol      0 Dec 10 16:01 .thisIsHidden
```

O arquivo `.thisIsHidden` está oculto simplesmente porque seu nome começa com `.`.

Porém, os diretórios `.` e `..` são especiais. `.` é um ponteiro para o diretório atual, enquanto `..` é um ponteiro para o diretório pai (o diretório que contém o diretório atual). No Linux, todo diretório contém pelo menos esses dois diretórios especiais.

TIP É possível combinar diversos parâmetros no `ls` (e em muitos outros comandos do Linux). `ls -l -a`, por exemplo, pode ser escrito como `ls -la`.

Compreendendo os tipos de arquivos

Já mencionamos que a primeira letra da saída de `ls -l` descreve o tipo de arquivo. Os três tipos de arquivo mais comuns são:

`-` (arquivo normal)

Um arquivo pode conter qualquer tipo de dado. os arquivos podem ser modificados, movidos, copiados e excluídos.

`d` (diretório)

Um diretório contém outros arquivos e diretórios e ajuda a organizar o sistema de arquivos. Tecnicamente, os diretórios são um tipo especial de arquivo.

`l` (link simbólico)

Este "arquivo" é um apontador para outro arquivo ou diretório em outro local do sistema de arquivos.

Além desses, existem outros três tipos de arquivos que você deve pelo menos conhecer, mas que estão fora do escopo desta lição:

`b` (dispositivo de bloco)

Este arquivo representa um dispositivo virtual ou físico, como os discos ou outro tipo de dispositivo de armazenamento. Por exemplo, o primeiro disco rígido do sistema pode ser representado por `/dev/sda`.

`c` (dispositivo de caracteres)

Este arquivo representa um dispositivo virtual ou físico. Os terminais (como o terminal principal em `/dev/ttyS0`) e portas seriais são exemplos comuns de dispositivos de caracteres.

`s` (socket)

Os sockets são como "dutos" que transmitem informações entre dois programas.

WARNING	Não altere nenhuma das permissões dos dispositivos de bloco, de caracteres ou sockets, a menos que saiba o que está fazendo. Isso pode fazer com que o sistema pare de funcionar!

Entendendo as permissões

Na saída de `ls -l`, as permissões de arquivo são mostradas logo após o tipo de arquivo, na forma de três grupos de três caracteres cada, na ordem `r`, `w` e `x`. Eis o significado deles. Lembre-se de que um hífen `-` representa a ausência de uma permissão determinada.

Permissões de arquivos

`r`

Representa *read* (leitura) e tem valor octal `4` (não se preocupe, falaremos de octais daqui a pouquinho). Significa a permissão de abrir um arquivo e ler seu conteúdo.

`w`

Representa *write* (escrita) e tem valor octal `2`. Significa a permissão de editar ou remover um arquivo.

`x`

Representa *execute* (execução) e tem valor octal `1`. Significa que o arquivo pode ser rodado como executável ou script.

Então, por exemplo, um arquivo com permissões `rw-` pode ser lido e escrito, mas não pode ser executado.

Permissões em diretórios

`r`

Representa *read* (leitura) e tem valor octal `4`. Significa a permissão para ler o conteúdo do diretório, como os nomes de arquivos. Mas não implica na permissão para ler os arquivos em si.

`w`

Representa *write* (escrita) e tem valor octal `2`. Significa a permissão de editar ou remover um arquivo em um diretório, além de alterar seu nome, permissões e proprietários. Se um usuário tiver permissão de escrita em um diretório, ele pode alterar as permissões de qualquer arquivo desse diretório, mesmo que não tenha permissões sobre o arquivo ou que ele pertença a um outro usuário.

`x`

Representa *execute* (execução) e tem valor octal `1`. Significa a permissão de entrar em um diretório, mas não de listar seus arquivos (para isso, é necessária a permissão `r`).

A última parte sobre os diretórios é um pouco confusa. Vamos imaginar, por exemplo, que temos um diretório chamado `Another_Directory`, com as seguintes permissões:

```
$ ls -ld Another_Directory/
d--xr-xr-x 2 carol carol 4,0K Dec 20 18:46 Another_Directory
```

Imagine também que dentro desse diretório tem um script do shell chamado `hello.sh` com as seguintes permissões:

```
-rwxr-xr-x 1 carol carol 33 Dec 20 18:46 hello.sh
```

Se você for a usuária `carol` e tentar listar o conteúdo de `Another_Directory`, aparecerá uma mensagem de erro, pois seu usuário não tem permissão de leitura para aquele diretório:

```
$ ls -l Another_Directory/
ls: cannot open directory 'Another_Directory/': Permission denied
```

Todavia, a usuária `carol` *tem* permissões de execução, o que significa que ela pode entrar no diretório. Assim, a usuária `carol` pode acessar arquivos dentro do diretório, desde que tenha as permissões corretas *para o arquivo respectivo*. Neste exemplo, a usuária tem permissões totais para o script `hello.sh`, então ela *pode* executar o script, embora *não possa* ler o conteúdo do diretório que o contém. Basta dispor do nome de arquivo completo.

```
$ sh Another_Directory/hello.sh
Hello LPI World!
```

Como dissemos anteriormente, as permissões são especificadas em sequência: primeiro para o proprietário do arquivo, depois para o grupo proprietário e por fim para outros usuários. Sempre que alguém tenta executar uma ação no arquivo, as permissões são verificadas da mesma maneira. Primeiro, o sistema verifica se o usuário atual possui o arquivo e, se for o caso, aplica apenas o primeiro conjunto de permissões. Caso contrário, ele verifica se o usuário atual pertence ao grupo que possui o arquivo. Nesse caso, ele aplica apenas o segundo conjunto de permissões. Em qualquer outro caso, o sistema aplicará o terceiro conjunto de permissões. Isso significa que, se o usuário atual for o proprietário do arquivo, apenas as permissões do proprietário serão efetivas, mesmo que as permissões do grupo ou outras permissões sejam mais permissivas do que as permissões do proprietário.

Modificando as permissões de arquivo

O comando `chmod` é usado para modificar as permissões de um arquivo e emprega pelo menos dois parâmetros: o primeiro descreve quais permissões devem ser alteradas e o segundo aponta para o arquivo ou diretório em que a alteração será feita. No entanto, as permissões para alterações podem ser descritas de duas maneiras, ou "modos", diferentes.

O primeiro, chamado *modo simbólico*, oferece um controle refinado, permitindo adicionar ou revogar uma única permissão sem modificar as de outras pessoas no conjunto de permissões. O outro modo, chamado *modo numérico*, é mais fácil de lembrar e mais rápido de usar quando queremos definir todos os valores de permissão de uma só vez.

Ambos os modos permitem o mesmo resultado final. Então, por exemplo, os comandos:

```
$ chmod ug+rw-x,o-rwx text.txt
```

e

```
$ chmod 660 text.txt
```

produzirão exatamente a mesma saída, um arquivo com as permissões definidas:

```
-rw-rw---- 1 carol carol  765 Dec 20 21:25 text.txt
```

Agora, vamos entender como cada modo funciona.

Modo simbólico

Ao descrever quais permissões devem ser alteradas no *modo simbólico*, o(s) primeiro(s) caractere(s) indicam as permissões que serão alteradas: permissões do usuário (`u`), do grupo (`g`), outras (`o`) e/ou as três juntas (`a`).

Em seguida, você precisa dizer ao comando o que fazer: você pode conceder uma permissão (`+`), revogar uma permissão (`-`) ou configurá-la para um valor específico (`=`).

Por fim, você especifica qual permissão deseja afetar: leitura (`r`), escrita (`w`) ou execução (`x`).

Por exemplo, imagine que temos um arquivo chamado `text.txt` com a seguinte permissão definida:

```
$ ls -l text.txt
-rw-r--r-- 1 carol carol 765 Dec 20 21:25 text.txt
```

Se quisermos conceder permissões de escrita aos membros do grupo proprietário arquivo, usamos o parâmetro `g+w`. Fica mais fácil entender assim: “Para o grupo (`g`), conceda (`+`) permissões de escrita (`w`)”. Assim, o comando ficaria:

```
$ chmod g+w text.txt
```

Vamos conferir o resultado com `ls`:

```
$ ls -l text.txt
-rw-rw-r-- 1 carol carol 765 Dec 20 21:25 text.txt
```

Se quisermos remover as permissões de leitura para o proprietário do mesmo arquivo, podemos raciocinar assim: “Para o usuário (`u`), revogue (`-`) as permissões de leitura (`r`)”. Assim, o parâmetro seria `u-r`, desta maneira:

```
$ chmod u-r text.txt
$ ls -l text.txt
--w-rw-r-- 1 carol carol 765 Dec 20 21:25 text.txt
```

E se quisermos definir as permissões exatamente como `rw-` para todos? Se for o caso, pensamos desta maneira: “Para todos (`a`), defina exatamente (`=`) leitura (`r`), escrita (`w`), mas não execução (`-`)”. Assim:

```
$ chmod a=rw- text.txt
$ ls -l text.txt
-rw-rw-rw- 1 carol carol 765 Dec 20 21:25 text.txt
```

Obviamente, é possível modificar várias permissões ao mesmo tempo. Nesse caso, nós as separamos com vírgula (`,`):

```
$ chmod u+rwx,g-x text.txt
$ ls -lh text.txt
-rwxrw-rw- 1 carol carol 765 Dec 20 21:25 text.txt
```

O exemplo acima pode ser lido como: “Para o usuário (`u`), conceda (`+`) permissões de leitura, escrita e execução (`rwx`), para o grupo (`g`) revogue (`-`) permissões de execução (`x`)”.

Quando executado em um diretório, `chmod` modifica apenas as permissões do diretório. `chmod` tem um modo recursivo, útil quando desejamos alterar as permissões para “todos os arquivos dentro de um diretório e seus subdiretórios”. Para usá-lo, adicione o parâmetro `-R` após o nome do comando e antes das permissões a alterar, da seguinte maneira:

```
$ chmod -R u+rwx Another_Directory/
```

Esse comando pode ser lido como: “Recursivamente (`-R`), para o usuário (`u`), conceda (`+`) permissões de leitura, escrita e execução (`rwx`)”.

WARNING

Tenha cautela e pense duas vezes antes de usar a opção `-R`, pois é fácil até demais alterar as permissões de arquivos e diretórios que não desejaríamos alterar, especialmente em diretórios com um grande número de arquivos e

subdiretórios.

Modo numérico

No *modo numérico*, as permissões são especificadas de maneira diferente: como um valor numérico de três dígitos na notação octal, um sistema numérico de base 8.

Cada permissão tem um valor correspondente e elas são especificadas na seguinte ordem: primeiro vem leitura (`r`), que é `4`, depois escrita (`w`), que é `2`, e por fim a execução (`x`), representada por `1`. Se não houver permissão, use o valor zero (`0`). Portanto, uma permissão de `rwx` seria `7` (`4+2+1`) e `r-x` seria `5` (`4+0+1`).

O primeiro dos três dígitos no conjunto de permissões representa as permissões do usuário (`u`), o segundo as do grupo (`g`) e o terceiro as do proprietário (`o`). Se desejássemos definir as permissões de um arquivo como `rw-rw----`, o valor octal seria `660`:

```
$ chmod 660 text.txt
$ ls -l text.txt
-rw-rw---- 1 carol carol 765 Dec 20 21:25 text.txt
```

Além disso, a sintaxe no *modo numérico* é a mesma do *modo simbólico*: o primeiro parâmetro representa as permissões que desejamos alterar e o segundo aponta para o arquivo ou diretório em que a alteração será feita.

TIP Se o valor de uma permissão for *ímpar*, é certeza que o arquivo é executável!

Qual sintaxe usar? O *modo numérico* é recomendado quando queremos alterar as permissões para um valor específico, por exemplo `640` (`rw- r-- ---`).

O *modo simbólico* é mais útil para alterar apenas um valor específico, quaisquer que sejam as permissões atuais do arquivo. Por exemplo, posso adicionar permissões de execução para o usuário somplesmente com `chmod u+x script.sh` sem me preocupar e nem sequer tocar nas permissões atuais do grupo e outras.

Modificando o proprietário de um arquivo

O comando `chown` é usado para modificar o proprietário de um arquivo ou diretório. A sintaxe é bastante simples:

```
chown username:groupname filename
```

Por exemplo, vamos verificar um arquivo chamado `text.txt`:

```
$ ls -l text.txt
-rw-rw---- 1 carol carol 1881 Dec 10 15:57 text.txt
```

O usuário que possui o arquivo é `carol` e o grupo também é `carol`. Agora, vamos modificar o grupo proprietário do arquivo para outro grupo, como `students`:

```
$ chown carol:students text.txt
$ ls -l text.txt
-rw-rw---- 1 carol students 1881 Dec 10 15:57 text.txt
```

Lembre-se de que o usuário que possui um arquivo não precisa pertencer ao grupo que o possui. No exemplo acima, a usuária `carol` não precisa ser membro do grupo `Students`. No entanto, ela precisa ser membro do grupo para poder transferir a propriedade do arquivo para ele.

O usuário ou o grupo podem ser omitidos se você não quiser alterá-los. Assim, para alterar apenas o grupo que possui um arquivo, usaríamos `chown :students text.txt`. Para mudar apenas o usuário, o comando seria `chown carol text.txt`. Como alternativa, você pode usar o comando `chgrp students text.txt` para alterar apenas o grupo.

A menos que você seja o administrador do sistema (root), não poderá alterar a propriedade de um arquivo pertencente a outro usuário ou grupo ao qual você não pertença. Se tentar fazer isso, receberá a mensagem de erro `Operation not permitted` (Operação não permitida).

Consultando os grupos

Antes de alterar a propriedade de um arquivo, pode ser útil saber quais grupos existem no sistema, quais usuários são membros de um grupo e a quais grupos um usuário pertence. Essas tarefas podem ser realizadas com dois comandos, `groups` e `groupmems`.

Para ver quais grupos existem no seu sistema, basta digitar `groups`:

```
$ groups
carol students cdrom sudo dip plugdev lpadmin sambashare
```

E para saber a quais grupos um usuário pertence, adicione o nome de usuário como parâmetro:

```
$ groups carol
```

```
carol : carol students cdrom sudo dip plugdev lpadmin sambashare
```

Para fazer o inverso, exibindo quais usuários pertencem a um grupo, use `groupmems`. O parâmetro `-g` especifica o grupo e `-l` lista todos os seus membros:

```
$ sudo groupmems -g cdrom -l
carol
```

TIP `groupmems` só pode ser executado como root, o administrador do sistema. Se você não estiver atualmente logado como root, adicione `sudo` antes do comando.

Permissões especiais

Além das permissões de leitura, escrita e execução para usuários, grupos e outros, cada arquivo pode ter três outras *permissões especiais* capazes de alterar a maneira como um diretório funciona ou como um programa é executado. Elas podem ser especificadas em modo simbólico ou numérico e são as seguintes:

Sticky Bit

O sticky bit, também chamado de *Restricted Deletion Flag*, tem o valor octal `1` e, no modo simbólico, é representado por um `t` dentro das permissões de outros. Aplica-se apenas aos diretórios e, no Linux, evita que os usuários removam ou renomeiem um arquivo em um diretório, a menos que sejam proprietários desse arquivo ou diretório.

Os diretórios com o sticky bit definido trazem um `t` no lugar do `x` nas permissões de *outros* na saída de `ls -l`:

```
$ ls -ld Sample_Directory/
drwxr-xr-t 2 carol carol 4096 Dec 20 18:46 Sample_Directory/
```

No modo numérico, as permissões especiais são especificadas usando uma "notação de 4 dígitos", sendo o primeiro dígito a permissão especial para agir. Por exemplo, para definir o sticky bit (valor `1`) para o diretório `Another_Directory` no modo numérico, com permissões `755`, o comando seria:

```
$ chmod 1755 Another_Directory
$ ls -ld Another_Directory
drwxr-xr-t 2 carol carol 4,0K Dec 20 18:46 Another_Directory
```

Set GID

O Set GID, também conhecido como SGID ou bit Set Group ID, tem o valor octal `2` e, no modo simbólico, é representado por um `s` nas permissões de *grupo*. Pode ser aplicado a arquivos ou diretórios executáveis. Nos arquivos executáveis, ele concede acesso aos privilégios do grupo que possui o arquivo ao processo resultante da execução do arquivo. Quando aplicado a diretórios, ele faz com que todos os arquivos ou diretórios criados com ele herdem o grupo do diretório pai.

Os arquivos e diretórios com o bit SGID trazem um `s` no lugar do `x` nas permissões de *grupo* na saída do `ls -l`:

```
$ ls -l test.sh
-rwxr-sr-x 1 carol carol 33 Dec 11 10:36 test.sh
```

Para adicionar permissões SGID a um arquivo no modo simbólico, o comando seria:

```
$ chmod g+s test.sh
$ ls -l test.sh
-rwxr-sr-x 1 carol root     33 Dec 11 10:36 test.sh
```

O exemplo a seguir ajudará a entender melhor os efeitos do SGID em um diretório. Suponha que tenhamos um diretório chamado `Sample_Directory`, pertencente ao usuário `carol` e ao grupo `users`, com a seguinte estrutura de permissões:

```
$ ls -ldh Sample_Directory/
drwxr-xr-x 2 carol users 4,0K Jan 18 17:06 Sample_Directory/
```

Agora, vamos mudar para este diretório e, usando o comando `touch`, criar um arquivo vazio dentro dele. O resultado seria:

```
$ cd Sample_Directory/
$ touch newfile
$ ls -lh newfile
-rw-r--r-- 1 carol carol 0 Jan 18 17:11 newfile
```

Como vemos, o arquivo pertence ao usuário `carol` e ao grupo `carol`. Mas, se o diretório tivesse a permissão SGID definida, o resultado seria diferente. Primeiro, vamos adicionar o bit SGID ao `Sample_Directory` e verificar os resultados:

```
$ sudo chmod g+s Sample_Directory/
$ ls -ldh Sample_Directory/
drwxr-sr-x 2 carol users 4,0K Jan 18 17:17 Sample_Directory/
```

O `s` nas permissões do grupo indica que o bit SGID está definido. Agora, vamos mudar para este diretório e, novamente, criar um arquivo vazio com o comando `touch`:

```
$ cd Sample_Directory/
$ touch emptyfile
$ ls -lh emptyfile
 -rw-r--r-- 1 carol users 0 Jan 18 17:20 emptyfile
```

Como vemos, o grupo que possui o arquivo é `users`. Isso ocorre porque o bit SGID fez o arquivo herdar o grupo proprietário do diretório pai, que é `users`.

Set UID

SUID, também conhecido como Set User ID, tem valor octal `4` e é representado por um `s` nas permissões de *usuário* no modo simbólico. Aplica-se apenas aos arquivos e seu comportamento é semelhante ao do bit SGID, mas o processo será executado com os privilégios do *usuário* proprietário do arquivo. Os arquivos com o bit SUID mostram um `s` no lugar do `x` nas permissões do usuário, na saída de `ls -l`:

```
$ ls -ld test.sh
-rwsr-xr-x 1 carol carol 33 Dec 11 10:36 test.sh
```

Podemos combinar diversas permissões especiais em um parâmetro somando-as. Assim, para definir o SGID (valor `2`) e o SUID (valor `4`) no modo numérico para o script `test.sh` com permissões `755`, digite:

```
$ chmod 6755 test.sh
```

E o resultado seria:

```
$ ls -lh test.sh
-rwsr-sr-x 1 carol carol 66 Jan 18 17:29 test.sh
```

TIP Se o seu terminal exibe cores, como é o caso da maioria atualmente, é fácil ver se

essas permissões especiais estão definidas olhando a saída de `ls -l`. Para o sticky bit, o nome do diretório pode ser mostrado em uma fonte preta com fundo azul. O mesmo se aplica aos arquivos com os bits SGID (fundo amarelo) e SUID (fundo vermelho). As cores podem ser diferentes dependendo da distribuição do Linux e das configurações do terminal que você usa.

Exercícios Guiados

1. Crie um diretório chamado `emptydir` usando o comando `mkdir emptydir`. Em seguida, usando `ls`, liste as permissões do diretório `emptydir`.

2. Crie um arquivo vazio chamado `emptyfile` com o comando `touch emptyfile`. Em seguida, usando `chmod` com notação simbólica, adicione permissões de execução ao proprietário do arquivo `emptyfile` e remova as permissões de gravação e execução para todos os outros. Faça isso usando apenas um comando `chmod`.

3. Quais serão as permissões de um arquivo chamado `text.txt` depois de você executar o comando `chmod 754 text.txt`?

4. Vamos supor que um arquivo chamado `test.sh` seja um script do shell com as seguintes permissões e propriedade:

```
-rwxr-sr-x 1 carol root     33 Dec 11 10:36 test.sh
```

   - Quais são as permissões do proprietário do arquivo?

   - Se o usuário `john` executar esse script, sob quais privilégios de usuário ele será executado?

   - Usando a notação numérica, qual deve ser a sintaxe do chmod para “desfazer” a permissão especial concedida a este arquivo?

5. Considere este arquivo:

```
$ ls -l /dev/sdb1
brw-rw---- 1 root disk 8, 17 Dec 21 18:51 /dev/sdb1
```

   Que tipo de arquivo é `sdb1`? E quem pode escrever nele?

6. Considere os 4 arquivos a seguir:

```
drwxr-xr-t 2 carol carol 4,0K Dec 20 18:46 Another_Directory
----r--r-- 1 carol carol    0 Dec 11 10:55 foo.bar
-rw-rw-r-- 1 carol carol 1,2G Dec 20 18:22 HugeFile.zip
drwxr-sr-x 2 carol users 4,0K Jan 18 17:26 Sample_Directory
```

Escreva as permissões correspondentes a cada arquivo e diretório usando a notação numérica de 4 dígitos.

`Another_Directory`

`foo.bar`

`HugeFile.zip`

`Sample_Directory`

Exercícios Exploratórios

1. Experimente o seguinte em um terminal: crie um arquivo vazio chamado `emptyfile` com o comando `touch emptyfile`. Agora "zere" as permissões do arquivo com `chmod 000 emptyfile`. O que acontecerá se você mudar as permissões de `emptyfile` passando apenas *um* valor de `chmod` para o modo numérico, como `chmod 4 emptyfile`? E se usarmos dois, como `chmod 44 emptyfile`? O que aprendemos sobre a maneira como `chmod` interpreta o valor numérico?

2. É possível executar um arquivo para o qual você tem permissões de execução, mas não de leitura (`--x`)? Por que ou porque não?

3. Considere as permissões do diretório temporário em um sistema Linux, `/tmp`:

```
$ ls -l /tmp
drwxrwxrwt  19 root root  16K Dec 21 18:58 tmp
```

   Usuário, grupo e outros têm permissões totais. Mas um usuário regular pode excluir *qualquer* arquivo dentro deste diretório? Por quê?

4. Um arquivo chamado `test.sh` tem as seguintes permissões: `-rwsr-xr-x`, o que indica que o bit SUID foi definido. Agora, execute os seguintes comandos:

```
$ chmod u-x test.sh
$ ls -l test.sh
-rwSr-xr-x 1 carol carol 33 Dec 11 10:36 test.sh
```

   O que nós fizemos? O que significa o `S` maiúsculo?

5. Como criar um diretório chamado `Box` no qual todos os arquivos pertençam automaticamente ao grupo `users` e só possam ser removidos pelo usuário que os criou?

Resumo

Por ser um sistema multiusuário, o Linux precisa de uma maneira de rastrear quem possui e quem pode acessar cada arquivo. Isso é feito por meio de um sistema de permissões em três níveis. Nesta lição, aprendemos como esse sistema funciona.

Nesta lição, você aprendeu como usar `ls` para obter informações sobre as permissões de arquivos, como controlar ou alterar quem pode criar, excluir ou modificar um arquivo com `chmod` nas notações *numérica* e *simbólica* e como alterar a propriedade de arquivos com `chown` e `chgrp`.

Os seguintes comandos foram discutidos nesta lição:

`ls`
: Lista os arquivos, incluindo opcionalmente detalhes como permissões.

`chmod`
: Altera as permissões de um arquivo ou diretório..

`chown`
: Altera o usuário e/ou o grupo proprietário de um arquivo ou diretório.

`chgrp`
: Altera o grupo proprietário de um arquivo ou diretório.

Respostas aos Exercícios Guiados

1. Crie um diretório chamado `emptydir` usando o comando `mkdir emptydir`. Em seguida, usando `ls`, liste as permissões do diretório `emptydir`.

   Adicionamos o parâmetro `-d` a `ls` para ver os atributos de arquivo de um diretório, em vez de listar seu conteúdo. Assim, a resposta é:

   ```
   ls -l -d emptydir
   ```

   Você ganha pontos extras se tiver juntado os dois parâmetros em um só, como em `ls -ld emptydir`.

2. Crie um arquivo vazio chamado `emptyfile` com o comando `touch emptyfile`. Em seguida, usando `chmod` com notação simbólica, adicione permissões de execução ao proprietário do arquivo `emptyfile` e remova as permissões de gravação e execução para todos os outros. Faça isso usando apenas um comando `chmod`.

   Pense desta maneira:

   - "Para o usuário que possui o arquivo (`u`) adicione (`+`) permissões de execução (`x`)", portanto `u+x`.
   - "Para o grupo (`g`) e outros usuários (`o`), remova (`-`) permissões de escrita (`w`) e execução (`x`)", portanto `go-wx`.

     Para combinar esses dois conjuntos de permissões, adicionamos uma virgula entre eles. Assim, o resultado final será:

     ```
     chmod u+x,go-wx emptyfile
     ```

3. Quais serão as permissões de um arquivo chamado `text.txt` depois de você executar o comando `chmod 754 text.txt`?

   Lembre-se de que em notação numérica cada dígito representa um conjunto de três permissões, cada uma com um valor respectivo: *leitura* é `4`, *escrita* é `2`, *execução* é `1` e a ausência de permissões é `0`. Obtemos o valor de um dígito somando os valores correspondentes de cada permissão. `7` é `4+2+1`, ou `rwx`, `5` é `4+0+1`, ou `r-x`, e `4` é somente leitura, ou `r--`. As permissões de `text.txt` seriam

```
rwxr-xr--
```

4. Vamos supor que um arquivo chamado `test.sh` seja um script do shell com as seguintes permissões e propriedade:

```
-rwxr-sr-x 1 carol root     33 Dec 11 10:36 test.sh
```

   - Quais são as permissões do proprietário do arquivo?

     As permissões do proprietário (2º ao 4 caracteres na saída de `ls -l`) são `rwx`, por isso a resposta é: "ler, escrever e executar o arquivo".

   - Se o usuário `john` executar esse script, sob quais privilégios de usuário ele será executado?

     Preste atenção às permissões do *grupo*. Elas são `r-s`, o que significa que o bit SGID está definido. O grupo proprietário do arquivo é `root`, de modo que o script, mesmo quando iniciado por um usuário regular, será executado com privilégios de root.

   - Usando a notação numérica, qual deve ser a sintaxe do chmod para "desfazer" a permissão especial concedida a este arquivo?

     Podemos "desfazer" as permissões especiais passando um 4º dígito, `0`, para `chmod`. As permissões atuais são `755`, então o comando deve ser `chmod 0755`.

5. Considere este arquivo:

```
$ ls -l /dev/sdb1
brw-rw---- 1 root disk 8, 17 Dec 21 18:51 /dev/sdb1
```

   Que tipo de arquivo é `sdb1`? E quem pode escrever nele?

   O primeiro caractere da saída de `ls -l` mostra o tipo de arquivo. `b` é um *dispositivo de bloco*, normalmente um disco (interno ou externo), conectado à máquina. O proprietário (`root`) e todos os usuários do grupo `disk` podem escrever nele.

6. Considere os 4 arquivos a seguir:

```
drwxr-xr-t 2 carol carol 4,0K Dec 20 18:46 Another_Directory
----r--r-- 1 carol carol    0 Dec 11 10:55 foo.bar
-rw-rw-r-- 1 carol carol 1,2G Dec 20 18:22 HugeFile.zip
```

```
drwxr-sr-x 2 carol users 4,0K Jan 18 17:26 Sample_Directory
```

Escreva as permissões correspondentes a cada arquivo e diretório usando a notação numérica de 4 dígitos.

As permissões correspondentes, em notação numérica, são as seguintes:

`Another_Directory`

Resposta: `1755`

`1` para o sticky bit, `755` para as permissões regulares (`rwx` para o usuário, `r-x` para grupo e outros).

`foo.bar`

Resposta: `0044`

Sem permissões especiais (de modo que o primeiro dígito é `0`), sem permissões para o usuário (`---`) e somente leitura (`r—r--`) para grupo e outros.

`HugeFile.zip`

Resposta: `0664` +Sem permissões especiais, de modo que o primeiro dígito é `0`. `6` (`rw-`) para usuário e grupo, `4` (`r--`) para outros.

`Sample_Directory`

Resposta: `2755`

`2` para o bit SGID, `7` (`rwx`) para o usuário, `5` (`r-x`) para o grupo e outros.

Respostas aos Exercícios Exploratórios

1. Experimente o seguinte em um terminal: crie um arquivo vazio chamado `emptyfile` com o comando `touch emptyfile`. Agora "zere" as permissões do arquivo com `chmod 000 emptyfile`. O que acontecerá se você mudar as permissões de `emptyfile` passando apenas *um* valor de `chmod` para o modo numérico, como `chmod 4 emptyfile`? E se usarmos dois, como `chmod 44 emptyfile`? O que aprendemos sobre a maneira como `chmod` interpreta o valor numérico?

   Lembre-se de que "zeramos" as permissões de `emptyfile`. Assim, seu estado inicial seria:

   ```
   ---------- 1 carol carol    0 Dec 11 10:55 emptyfile
   ```

   Agora, vamos tentar o primeiro comando, `chmod 4 emptyfile`:

   ```
   $ chmod 4 emptyfile
   $ ls -l emptyfile
   -------r-- 1 carol carol 0 Dec 11 10:55 emptyfile
   ```

   As permissões de *outros* foram alteradas. E se usássemos dois dígitos, como em `chmod 44 emptyfile`?

   ```
   $ chmod 44 emptyfile
   $ ls -l emptyfile
   ----r--r-- 1 carol carol 0 Dec 11 10:55 emptyfile
   ```

   Agora, as permissões de *grupo* e *outros* foram afetadas. A partir daí, podemos concluir que em notação numérica, `chmod` lê o valor "de trás pra frente", do dígito menos significativo (*others*) para o mais significativo (*usuário*). Se você passar um só dígito, as permissões de *outros* são modificadas. Com dois dígitos modificamos *grupo* e *outros*, com três modificamos *usuário*, *grupo* e *outros* e com quatro modificamos *usuário*, *grupo*, *outros* e as permissões especiais.

2. É possível executar um arquivo para o qual você tem permissões de execução, mas não de leitura (`--x`)? Por que ou porque não?

   No início, a resposta parece óbvia: se você tem permissão de execução, o arquivo deveria poder ser executado. Isso se aplica aos programas em formato binário que são executados diretamente pelo kernel. No entanto, existem programas (por exemplo, scripts do shell) que devem antes ser lidos e interpretados. Nesses casos, a permissão de leitura (`r`) também deve ser

configurada.

3. Considere as permissões do diretório temporário em um sistema Linux, `/tmp`:

```
$ ls -l /tmp
drwxrwxrwt  19 root root  16K Dec 21 18:58 tmp
```

Usuário, grupo e outros têm permissões totais. Mas um usuário regular pode excluir *qualquer* arquivo dentro deste diretório? Por quê?

`/tmp` é o que chamamos um diretório *escrito por todos*, ou seja, qualquer usuário pode escrever nele. Mas não queremos que um usuário modifique arquivos criados por outros, por isso o *sticky bit* foi definido (como indicado pelo `t` nas permissões de *outros*). Graças a isso, um usuário pode excluir arquivos em `/tmp`, mas somente se tiver criado esses arquivos.

4. . Um arquivo chamado `test.sh` tem as seguintes permissões: `-rwsr-xr-x`, o que indica que o bit SUID foi definido. Agora, execute os seguintes comandos:

```
$ chmod u-x test.sh
$ ls -l test.sh
-rwSr-xr-x 1 carol carol 33 Dec 11 10:36 test.sh
```

O que nós fizemos? O que significa o `S` maiúsculo?

Removemos as permissões de execução do usuário dono do arquivo. O `s` (ou `t`) toma o lugar do `x` na saída de `ls -l`, por isso o sistema precisa de uma maneira de mostrar se o usuário tem permissões de execução ou não. Para isso, ele muda a caixa do caractere especial.

Um `s` minúsculo no primeiro grupo de permissões indica que o usuário dono do arquivo tem permissões de execução e que o bit SUID foi definido. Um `S` maiúsculo indica que o usuário dono do arquivo não tem (`-`) permissões de execução e que o bit SUID foi definido.

Pode-se dizer o mesmo do SGID. Um `s` minúsculo no segundo grupo de permissões indica que o grupo dono do arquivo tem permissões de execução e que o bit SUID foi definido. Um `S` maiúsculo indica que o grupo dono do arquivo não tem (`-`) permissões de execução e que o bit SUID foi definido.

Isso também vale para o sticky bit, representado pelo `t` No terceiro grupo de permissões. O `t` minúsculo indica que o sticky bit foi definido e que os outros têm permissões de execução. O `T` maiúsculo indica que o sticky bit foi definido e que os outros não têm permissões de execução.

5. Como criar um diretório chamado `Box` no qual todos os arquivos pertençam automaticamente ao grupo `users` e só possam ser removidos pelo usuário que os criou?

   Este seria um processo de vários passos. O primeiro é criar o diretório:

   ```
   $ mkdir Box
   ```

   Queremos que cada arquivo criado dentro deste diretório seja atribuído automaticamente ao grupo `users`. Para isso, configuramos esse grupo como proprietário do diretório, depois definimos nele o bit SGID. Também precisamos garantir que qualquer membro do grupo possa escrever nesse diretório.

   Como não precisamos nos importar com as outras permissões e queremos apenas "ligar" os bits especiais, faz sentido usar o modo simbólico:

   ```
   $ chown :users Box/
   $ chmod g+wxs Box/
   ```

   Note que se seu usuário atual não pertencer ao grupo `users`, será necessário usar o comando `sudo` antes dos comandos acima para fazer a alteração como root.

   Para terminar, vamos garantir que somente o usuário criador de um arquivo tem a permissão de excluí-lo. Para isso, definimos o sticky bit (representado por um `t`) no diretório. Lembre-se de que ele é configurado nas permissões de outros (`o`).

   ```
   $ chmod o+t Box/
   ```

   As permissões do diretório `Box` devem aparecer da seguinte maneira:

   ```
   drwxrwsr-t 2 carol users  4,0K Jan 18 19:09 Box
   ```

   Claro que também é possível especificar o bit SGID e o sticky bit usando somente um comando `chmod`:

   ```
   $ chmod g+wxs,o+t Box/
   ```

   Se você pensou nisso, parabéns!

5.4 Diretórios e arquivos especiais

Referência ao LPI objectivo

Linux Essentials v1.6, Exam 010, Objective 5.4

Peso

1

Áreas chave de conhecimento

- Usando arquivos e diretórios temporários
- Links simbólicos

Segue uma lista parcial de arquivos, termos e utilitários utilizados

- `/tmp/`, `/var/tmp/` and Sticky Bit
- `ls -d`
- `ln -s`

5.4 Lição 1

Certificação:	Linux Essentials
Versão:	1.6
Tópico:	5 Segurança e permissões de arquivos
Objetivo:	5.4 Diretórios e arquivos especiais
Lição:	1 de 1

Introdução

No Linux, tudo é tratado como um arquivo. No entanto, alguns arquivos recebem um tratamento especial, seja devido ao local em que estão armazenados, como os arquivos temporários, ou à maneira como interagem com o sistema de arquivos, como os links. Nesta lição, aprenderemos onde esses arquivos estão localizados, como funcionam e como gerenciá-los.

Arquivos temporários

Os arquivos temporários são arquivos usados pelos programas para armazenar dados que serão necessários apenas por um curto período de tempo — por exemplo, dados de processos em execução, logs de falhas, arquivos de rascunho de um salvamento automático, arquivos intermediários usados durante uma conversão, arquivos de cache e assim por diante.

Localização dos arquivos temporários

A versão 3.0 do *Filesystem Hierarchy Standard* (FHS) define locais padrão para os arquivos temporários nos sistemas Linux. Cada local tem uma finalidade e um comportamento diferentes, e é recomendável que os desenvolvedores sigam as convenções definidas pelo FHS ao gravar dados

temporários no disco.

`/tmp`

De acordo com o FHS, não se deve pressupor que os arquivos escritos aqui serão preservados entre as invocações de um programa. A *recomendação* é que esse diretório seja limpo (todos os arquivos apagados) durante a inicialização do sistema, embora isso *não seja obrigatório*.

`/var/tmp`

Outro local para arquivos temporários, mas este *não deve ser limpo* durante a inicialização do sistema, ou seja, os arquivos armazenados aqui geralmente persistem entre as reinicializações.

`/run`

Este diretório contém arquivos variáveis de tempo de execução usados pelos processos ativos, como os arquivos identificadores de processo (`.pid`). Os programas que precisam de mais de um arquivo de tempo de execução podem criar subdiretórios aqui. Este local *deve ser limpo* durante a inicialização do sistema. Antigamente essa finalidade pertencia a `/var/run` e, em alguns sistemas, `/var/run` pode ser um link simbólico para `/run`.

Observe que nada impede o programa de criar arquivos temporários em outro local do sistema, mas é recomendável respeitar as convenções definidas pelo FHS.

Permissões em arquivos temporários

A existência de diretórios temporários que contemplam todo o sistema em um sistema multiusuário apresenta alguns desafios em relação às permissões de acesso. A princípio, poderíamos pensar que esses diretórios seriam “graváveis por todos”, ou seja, qualquer usuário poderia escrever ou excluir dados dele. Mas se assim fosse, como impedir que um usuário apagasse ou modificasse arquivos criados por outro?

A solução é uma permissão especial chamada *sticky bit*, que se aplica a diretórios e arquivos. Porém, por razões de segurança, o kernel do Linux ignora o sticky bit quando ele é aplicado a arquivos. Quando esse bit especial é definido para um diretório, ele impede que os usuários removam ou renomeiem um arquivo nesse diretório, a menos que sejam proprietários do arquivo.

Os diretórios com o sticky bit definido mostram um `t` no lugar do `x` nas permissões de *outros* na saída de `ls -l`. Por exemplo, vamos verificar as permissões dos diretórios `/tmp` e `/var/tmp`:

```
$ ls -ldh /tmp/ /var/tmp/
drwxrwxrwt 25 root root 4,0K Jun  7 18:52 /tmp/
```

```
drwxrwxrwt 16 root root 4,0K Jun  7 09:15 /var/tmp/
```

Como você pode ver pelo `t` no lugar do `x` na permissão para *outros*, os dois diretórios estão com o sticky bit definido.

Para definir o sticky bit em um diretório usando `chmod` no modo numérico, use a notação de quatro dígitos, sendo `1` o primeiro dígito. Por exemplo:

```
$ chmod 1755 temp
```

define o sticky bit para o diretório chamado `temp` e suas permissões como `rwxr-xr-t`.

Ao usar o modo simbólico, empregue o parâmetro `t`. Assim, `+t` para definir o sticky bit e `-t` para desativá-lo. Desta maneira:

```
$ chmod +t temp
```

Compreendendo os links

Já dissemos que, no Linux, tudo é tratado como um arquivo. Mas existe um tipo *especial* de arquivo chamado *link*. Há dois tipos de links em um sistema Linux:

Links simbólicos

Também chamados de *soft links*, eles apontam para o caminho de outro arquivo. Se excluirmos o arquivo para o qual o link aponta (chamado *destino*), o link ainda existirá, mas "para de funcionar", pois passará a apontar para "nada".

Links físicos

Um link físico é como como um segundo nome para o arquivo original. Eles *não* são duplicatas, mas sim uma entrada adicional no sistema de arquivos que aponta para o mesmo local (inode) no disco.

TIP

Um *inode* é uma estrutura de dados que armazena os atributos de um objeto (como um arquivo ou diretório) em um sistema de arquivos. Dentre esses atributos estão o nome de arquivo, permissões, proprietário e os blocos do disco nos quais estão armazenados os dados referentes àquele objeto. São semelhantes a um verbete em um índice; por isso o nome, que vem de "index node".

Trabalhando com links físicos

Criando links físicos

O comando para criar um link físico no Linux é `ln`. A sintaxe básica é:

```
$ ln TARGET LINK_NAME
```

O `TARGET` já deve existir (trata-se do arquivo para o qual o link apontará) e, se o destino não estiver no diretório atual ou se você deseja criar o link em outro lugar, é *obrigatório* especificar o caminho completo para ele. Por exemplo, o comando

```
$ ln target.txt /home/carol/Documents/hardlink
```

Cria um arquivo chamado `hardlink` no diretório `/home/carol/Documents/`, vinculado ao arquivo `target.txt` no diretório atual.

Se deixarmos de fora o último parâmetro (`LINK_NAME`), será criado um link com o mesmo nome do destino no diretório atual.

Gerenciando os links físicos

Os links físicos são entradas no sistema de arquivos que têm nomes diferentes, mas que apontam para os mesmos dados no disco. Todos esses nomes são equivalentes e podem ser usados para se referir a um arquivo. Se você alterar o conteúdo de um dos nomes, o conteúdo de todos os outros nomes que apontam para aquele arquivo será alterado, já que todos apontam para os mesmos dados. Se você excluir um dos nomes, os outros nomes continuarão a funcionar.

Isso acontece porque quando você "exclui" um arquivo, os dados não são realmente apagados do disco. O sistema simplesmente exclui a entrada na tabela do sistema de arquivos apontando para o inode correspondente aos dados no disco. Mas se houver uma segunda entrada apontando para o mesmo inode, ainda será possível acessar os dados. É como se fossem duas estradas convergindo no mesmo ponto. Mesmo se bloquearmos ou redirecionarmos uma das estradas, ainda será possível chegar ao destino usando a outra.

Para conferir, podemos usar o parâmetro `-i` de `ls`. Considere o seguinte conteúdo de um diretório:

```
$ ls -li
total 224
```

```
3806696 -r--r--r-- 2 carol carol 111702 Jun  7 10:13 hardlink
3806696 -r--r--r-- 2 carol carol 111702 Jun  7 10:13 target.txt
```

O número anterior às permissões é o número inode. Percebeu que tanto o arquivo `hardlink` quanto o arquivo `target.txt` têm o mesmo número (`3806696`)? Isso ocorre porque um é o link físico do outro.

Mas qual é o original e qual é o link? Não dá pra dizer, já que, na prática, eles são a mesma coisa.

Observe que todo link físico que aponta para um arquivo aumenta a *contagem de links* do arquivo. Este é o número logo após as permissões na saída de `ls -l`. Por padrão, todo arquivo tem uma contagem de links de `1` (os diretórios têm uma contagem de `2`), e cada link físico apontando para ele aumenta a contagem em um. Por isso a contagem de links é `2` nos arquivos da lista acima.

Ao contrário dos links simbólicos, só é possível criar links físicos para arquivos, e tanto o link quanto o destino devem residir no mesmo sistema de arquivos.

Movendo e removendo links físicos

Como os links físicos são tratados como arquivos regulares, eles podem ser excluídos com `rm` e renomeados ou movidos no sistema de arquivos com `mv`. E como um link rígido aponta para o mesmo inode do destino, ele pode ser movido livremente, sem medo de "quebrar" o link.

Links simbólicos

Criando links simbólicos

O comando usado para criar um link simbólico também é `ln`, mas com o parâmetro `-s` adicionado. Assim:

```
$ ln -s target.txt /home/carol/Documents/softlink
```

Criamos assim um arquivo chamado `softlink` no diretório `/home/carol/Documents/`, apontando para o arquivo `target.txt` no diretório atual.

Como no caso dos links físicos, é possível omitir o nome do link para criar um link com o mesmo nome do destino no diretório atual.

Gerenciando links simbólicos

Os links simbólicos apontam para outro caminho no sistema de arquivos. Podemos criar links

simbólicos para arquivos *e* diretórios, mesmo em diferentes partições. É muito fácil identificar um link simbólico na saída de `ls`:

```
$ ls -lh
total 112K
-rw-r--r-- 1 carol carol 110K Jun  7 10:13 target.txt
lrwxrwxrwx 1 carol carol   12 Jun  7 10:14 softlink -> target.txt
```

No exemplo acima, o primeiro caractere nas permissões para o arquivo `softlink` é `l`, indicando um link simbólico. Além disso, logo após o nome do arquivo, vemos o nome do destino para o qual o link aponta, o arquivo `target.txt`.

Note que nas listagens de arquivos e diretórios os links simbólicos sempre mostram as permissões `rwx` para o usuário, o grupo e outros, mas, na prática, as permissões de acesso para eles são as mesmas do destino.

Movendo e removendo links simbólicos

Como no caso dos links físicos, os links simbólicos podem ser removidos usando `rm` e movidos ou renomeados com `mv`. No entanto, deve-se tomar cuidado especial ao criá-los, para evitar "quebrar" o link se ele for movido de seu local original.

Ao criar links simbólicos, é preciso estar ciente de que, a menos que um caminho seja totalmente especificado, o local do destino será interpretado como sendo *relativo* ao local do link. Isso pode criar problemas caso o link ou o arquivo para o qual ele aponta seja movido.

É mais fácil entender isso com um exemplo. Digamos que temos um arquivo chamado `original.txt` no diretório atual e desejamos criar um link simbólico para ele chamado `softlink`. Poderíamos usar:

```
$ ln -s original.txt softlink
```

E aparentemente tudo correria bem. Vamos verificar com `ls`:

```
$ ls -lh
total 112K
-r--r--r-- 1 carol carol 110K Jun  7 10:13 original.txt
lrwxrwxrwx 1 carol carol   12 Jun  7 19:23 softlink -> original.txt
```

Observe como o link é elaborado: `softlink` aponta para (→) `original.txt`. Entretanto, vamos

ver o que acontece se movermos o link para o diretório pai e tentarmos exibir seu conteúdo com o comando `less`:

```
$ mv softlink ../
$ less ../softlink
../softlink: No such file or directory
```

Como o caminho para `original.txt` não foi especificado, o sistema pressupõe que ele está no mesmo diretório que o link. Quando isso deixa de ser verdade, o link para de funcionar.

Para evitar isso, devemos sempre especificar o caminho completo para o destino ao criar o link:

```
$ ln -s /home/carol/Documents/original.txt softlink
```

Dessa forma, o link continuará a funcionar mesmo se for movido, porque aponta para a localização absoluta do destino. Verifique com `ls`:

```
$ ls -lh
total 112K
lrwxrwxrwx 1 carol carol   40 Jun  7 19:34 softlink -> /home/carol/Documents/original.txt
```

Exercícios Guiados

1. Imagine que um programa precisa criar um arquivo temporário de uso único, que não será novamente necessário após o encerramento do programa. Qual seria o diretório correto para criar esse arquivo?

2. Qual é o diretório temporário que *deve* ser limpo durante o processo de inicialização?

3. Qual é o parâmetro de `chmod` no modo *simbólico* para ativar o sticky bit em um diretório?

4. Imagine que existe um arquivo chamado `document.txt` no diretório `/home/carol/Documents`. Com qual comando criaríamos um link simbólico para ele chamado `text.txt` no diretório atual?

5. Explique a diferença entre um link físico para um arquivo e uma cópia desse arquivo.

Exercícios Exploratórios

1. Imagine que dentro de um diretório você cria um arquivo chamado `recipes.txt`. Dentro deste diretório, você também criará um link físico para este arquivo, chamado `receitas.txt`, e um link simbólico (ou *soft*) para este, chamado `rezepte.txt`.

   ```
   $ touch recipes.txt
   $ ln recipes.txt receitas.txt
   $ ln -s recipes.txt rezepte.txt
   ```

   O conteúdo do diretório deve aparecer assim:

   ```
   $ ls -lhi
   total 160K
   5388833 -rw-r--r-- 4 carol carol 77K jun 17 17:25 receitas.txt
   5388833 -rw-r--r-- 4 carol carol 77K jun 17 17:25 recipes.txt
   5388837 lrwxrwxrwx 1 carol carol  12 jun 24 10:12 rezepte.txt -> receitas.txt
   ```

   Lembre-se que, por ser um link físico, `receitas.txt` aponta para o mesmo inode que `recipes.txt`. O que aconteceria com o link `rezepte.txt` se o nome `receitas.txt` fosse excluído? Por quê?

2. Imagine que você tem um pendrive conectado ao sistema e montado em `/media/youruser/FlashA`. Você deseja criar em seu diretório pessoal um link chamado `schematics.pdf` apontando para o arquivo `esquema.pdf` no diretório raiz do pendrive. Então, você digita o comando:

   ```
   $ ln /media/youruser/FlashA/esquema.pdf ~/schematics.pdf
   ```

   O que aconteceria? Por quê?

3. Considere a seguinte saída de `ls -lah`:

   ```
   $ ls -lah
   total 3,1M
   drwxr-xr-x 2 carol carol 4,0K jun 17 17:27 .
   drwxr-xr-x 5 carol carol 4,0K jun 17 17:29 ..
   -rw-rw-r-- 1 carol carol 2,8M jun 17 15:45 compressed.zip
   -rw-r--r-- 4 carol carol  77K jun 17 17:25 document.txt
   -rw-rw-r-- 1 carol carol 216K jun 17 17:25 image.png
   ```

```
-rw-r--r-- 4 carol carol  77K jun 17 17:25 text.txt
```

- Quantos links apontam para o arquivo `document.txt`?
- Trata-se de links físicos ou simbólicos?
- Qual parâmetro você deveria passar para `ls` para ver qual inode é ocupado por cada arquivo?

4. Imagine que você tem, em seu diretório `~/Documents`, um arquivo chamado `clients.txt` contendo alguns nomes de clientes e um diretório chamado `somedir`. Dentro dele, existe um arquivo *diferente*, *também* chamado `clients.txt`, contendo nomes diferentes. Para replicar essa estrutura, use os seguintes comandos:

```
$ cd ~/Documents
$ echo "John, Michael, Bob" > clients.txt
$ mkdir somedir
$ echo "Bill, Luke, Karl" > somedir/clients.txt
```

Em seguida você cria um link dentro de `somedir` chamado `partners.txt` apontando para esse arquivo, com os comandos:

```
$ cd somedir/
$ ln -s clients.txt partners.txt
```

Assim, a estrutura do diretório é:

```
Documents
|-- clients.txt
`-- somedir
    |-- clients.txt
    `-- partners.txt -> clients.txt
```

Agora, você move `partners.txt` de `somedir` para `~/Documents`, e lista seu conteúdo.

```
$ cd ~/Documents/
```

```
$ mv somedir/partners.txt .
$ less partners.txt
```

O link ainda funciona? Se sim, qual arquivo terá seu conteúdo listado? Por quê?

5. Considere os arquivos a seguir:

```
-rw-r--r-- 1 carol carol 19 Jun 24 11:12 clients.txt
lrwxrwxrwx 1 carol carol 11 Jun 24 11:13 partners.txt -> clients.txt
```

Quais são as permissões de acesso de `partners.txt`? Por quê?

Resumo

Nesta lição, você aprendeu:

- Onde são armazenados os arquivos temporários.
- Qual a permissão especial aplicada a eles.
- O que são links.
- A diferença entre os links *simbólicos* e *físicos*.
- Como criar links.
- Como movê-los, renomeá-los ou excluí-los.

Os seguintes comandos foram discutidos nesta lição:

- `ln`
- O parâmetro `-i` de `ls`

Respostas aos Exercícios Guiados

1. Imagine que um programa precisa criar um arquivo temporário de uso único, que não será novamente necessário após o encerramento do programa. Qual seria o diretório correto para criar esse arquivo?

   Como não precisamos mais do arquivo depois que o programa for encerrado, o diretório correto é `/tmp`.

2. Qual é o diretório temporário que *deve* ser limpo durante o processo de inicialização?

   O diretório é `/run` ou, em certos sistemas, `/var/run`.

3. Qual é o parâmetro de `chmod` no modo *simbólico* para ativar o sticky bit em um diretório?

   O símbolo do sticky bit no modo simbólico é `t`. Como queremos habilitar (adicionar) essa permissão no diretório, o parâmetro deve ser `+t`.

4. Imagine que existe um arquivo chamado `document.txt` no diretório `/home/carol/Documents`. Com qual comando criaríamos um link simbólico para ele chamado `text.txt` no diretório atual?

   `ln -s` é o comando para criar um link simbólico. Como é necessário especificar o caminho completo do arquivo para o qual estamos criando o link, o comando seria:

   ```
   $ ln -s /home/carol/Documents/document.txt text.txt
   ```

5. Explique a diferença entre um link físico para um arquivo e uma cópia desse arquivo.

   Um link físico é apenas outro nome para um arquivo. Mesmo que pareça uma duplicata do arquivo original, para todos os efeitos, o link e o original são iguais, pois apontam para os mesmos dados no disco. As alterações feitas no conteúdo do link serão refletidas no original e vice-versa. Uma cópia é uma entidade completamente independente, ocupando um lugar diferente no disco. As alterações na cópia não serão refletidas no original e vice-versa.

Respostas aos Exercícios Exploratórios

1. Imagine que dentro de um diretório você crie um arquivo chamado `recipes.txt`. Dentro deste diretório, você também criará um link físico para este arquivo, chamado `receitas.txt`, e um link simbólico (ou *soft*) para este chamado `rezepte.txt`.

   ```
   $ touch recipes.txt
   $ ln recipes.txt receitas.txt
   $ ln -s recipes.txt rezepte.txt
   ```

   O conteúdo do diretório deve aparecer assim:

   ```
   $ ls -lhi
   total 160K
   5388833 -rw-r--r-- 4 carol carol 77K jun 17 17:25 receitas.txt
   5388833 -rw-r--r-- 4 carol carol 77K jun 17 17:25 recipes.txt
   5388837 lrwxrwxrwx 1 carol carol  12 jun 24 10:12 rezepte.txt -> recipes.txt
   ```

   Lembre-se que, por ser um link físico, `receitas.txt` aponta para o mesmo inode que `recipes.txt`. O que aconteceria com o link `rezepte.txt` se o nome `receitas.txt` fosse excluído? Por quê?

   O link simbólico `rezepte.txt` deixaria de funcionar, porque os links simbólicos apontam para nomes, e não inodes, e o nome `receitas.txt` não existe mais, mesmo que os dados ainda estejam no disco com o nome `recipes.txt`.

2. Imagine que você tem um pendrive conectado ao sistema e montado em `/media/youruser/FlashA`. Você deseja criar em seu diretório pessoal um link chamado `schematics.pdf` apontando para o arquivo `esquema.pdf` no diretório raiz do pendrive. Então, você digita o comando:

   ```
   $ ln /media/youruser/FlashA/esquema.pdf ~/schematics.pdf
   ```

   O que aconteceria? Por quê?

   O comando falharia. A mensagem de erro seria `Invalid cross-device link` (Link inválido entre dispositivos), esclarecendo o motivo: os links físicos não podem apontar para um destino em uma partição ou dispositivo diferente. A única maneira de criar um link como esse é usar um link *simbólico* ou *soft*, adicionando o parâmetro `-s` a `ln`.

3. Considere a seguinte saída de `ls -lah`:

```
$ ls -lah
total 3,1M
drwxr-xr-x 2 carol carol 4,0K jun 17 17:27 .
drwxr-xr-x 5 carol carol 4,0K jun 17 17:29 ..
-rw-rw-r-- 1 carol carol 2,8M jun 17 15:45 compressed.zip
-rw-r--r-- 4 carol carol  77K jun 17 17:25 document.txt
-rw-rw-r-- 1 carol carol 216K jun 17 17:25 image.png
-rw-r--r-- 4 carol carol  77K jun 17 17:25 text.txt
```

   - Quantos links apontam para o arquivo `document.txt`?

     Todos os arquivos começam com `1` na contagem de links. Como a contagem de links desse arquivo é `4`, existem três links apontando para ele.

   - Trata-se de links físicos ou simbólicos?

     São links físicos, já que os links simbólicos não aumentam a contagem de links de um arquivo.

   - Qual parâmetro você deveria passar para `ls` para ver qual inode é ocupado por cada arquivo?

     O parâmetro é `-i`. O inode será mostrado na primeira coluna da saída de `ls`, como abaixo:

```
$ ls -lahi
total 3,1M
5388773 drwxr-xr-x 2 rigues rigues 4,0K jun 17 17:27 .
5245554 drwxr-xr-x 5 rigues rigues 4,0K jun 17 17:29 ..
5388840 -rw-rw-r-- 1 rigues rigues 2,8M jun 17 15:45 compressed.zip
5388833 -rw-r--r-- 4 rigues rigues  77K jun 17 17:25 document.txt
5388837 -rw-rw-r-- 1 rigues rigues 216K jun 17 17:25 image.png
5388833 -rw-r--r-- 4 rigues rigues  77K jun 17 17:25 text.txt
```

4. Imagine que você tem, em seu diretório `~/Documents`, um arquivo chamado `clients.txt` contendo alguns nomes de clientes e um diretório chamado `somedir`. Dentro dele, existe um arquivo *diferente*, *também* chamado `clients.txt`, contendo nomes diferentes. Para replicar essa estrutura, use os seguintes comandos:

```
$ cd ~/Documents
```

```
$ echo "John, Michael, Bob" > clients.txt
$ mkdir somedir
$ echo "Bill, Luke, Karl" > somedir/clients.txt
```

Em seguida você cria um link dentro de `somedir` chamado `partners.txt` apontando para esse arquivo, com os comandos:

```
$ cd somedir/
$ ln -s clients.txt partners.txt
```

Assim, a estrutura do diretório é:

```
Documents
|-- clients.txt
`-- somedir
    |-- clients.txt
    `-- partners.txt -> clients.txt
```

Agora, você move `partners.txt` de `somedir` para `~/Documents`, e lista seu conteúdo.

```
$ cd ~/Documents/
$ mv somedir/partners.txt .
$ less partners.txt
```

O link ainda funciona? Se sim, qual arquivo terá seu conteúdo listado? Por quê?

Isto é quase uma “pegadinha”, mas o link funcionará e o arquivo listado será o que está em `~/Documents`, contendo os nomes `John`, `Michael`, `Bob`.

Lembre-se de que, como não especificamos o caminho completo para o alvo `clients.txt` ao criar o link simbólico `partners.txt`, o local do destino será interpretado como sendo relativo ao local do link, que neste caso é o diretório atual.

Quando o link é movido de `~/Documents/somedir` para `~/Documents`, ele deveria parar de funcionar, já que o destino não estava mais no mesmo diretório do link. Porém, coincidentemente existe um arquivo chamado `clients.txt` em `~/Documents`, de forma que o link apontará para esse arquivo em vez do destino original dentro de `~/somedir`.

Para evitar isso, sempre especifique o caminho completo para o destino ao criar um link simbólico.

5. Considere os arquivos a seguir:

```
-rw-r--r-- 1 rigues rigues 19 Jun 24 11:12 clients.txt
lrwxrwxrwx 1 rigues rigues 11 Jun 24 11:13 partners.txt -> clients.txt
```

   Quais são as permissões de acesso de `partners.txt`? Por quê?

   As permissões de acesso de `partners.txt` são `rw-r—r--`, já que os links sempre herdam as mesmas permissões de acesso do alvo.

Imprint

© 2023 by Linux Professional Institute: Materiais Didáticos, “Linux Essentials (Versão 1.6)”.

PDF gerado: 2023-04-14

Este trabalho está licenciado sob Creative Commons Licença Atribuição-NãoComercial-SemDerivações 4.0 Internacional (CC BY-NC-ND 4.0). Para ver uma cópia desta licença, visite

https://creativecommons.org/licenses/by-nc-nd/4.0/

Embora o Linux Professional Institute tenha agido de boa fé para garantir que as informações e instruções contidas neste trabalho sejam exatas, o Linux Professional Institute isenta-se de qualquer responsabilidade por erros ou omissões, incluindo, sem limitações, a responsabilidade por danos resultantes do uso ou confiança nesta obra. O uso das informações e instruções contidas neste trabalho deve ser feito por sua própria conta e risco. Se as amostras de código ou outras tecnologias contidas ou descritas neste trabalho estiverem sujeitas a licenças de código aberto ou direitos de propriedade intelectual de terceiros, é sua responsabilidade garantir que seu uso esteja em conformidade com tais licenças e/ou direitos.

Os Materiais Didáticos da LPI são uma iniciativa do Linux Professional Institute (https://lpi.org). Os Materiais Didáticos e suas traduções estão disponíveis em https://learning.lpi.org.

Para perguntas e comentários sobre esta edição, bem como sobre todo o projeto, escreva para: learning@lpi.org.